施耐德SoMachine控制器应用及编程指南

李幼涵　编著



机械工业出版社
CHINA MACHINE PRESS

VISIT…

LANZAROTE

Caliente.com

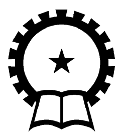


李幼涵，高级工程师。曾任职大型国企，参与的项目获得国家科技进步二等奖，个人获北京市优秀青年工程师奖，同年晋升高级工程师。1998年留学德国，获硕士学位，并在德国大众汽车公司从事设计工作两年。2001年回国加入施耐德电气公司，任职工业自动化运动控制部工程师、技术支持及培训经理，2011年晋职施耐德电气全球技术专家。

该作者从事自动控制设计、调试工作30多年，具备机械、电气综合设计，调试和组织、管理、实施能力，积累了丰富的理论和实践经验，为企业提供了各种控制问题的解决方案。在施耐德电气公司组织并完成了多个项目的设计及调试工作。出版专著《伺服运动控制系统的结构及应用》、《LEXIUM05伺服驱动器技术指南及案例》、合著《运动控制技术与应用》、合著《机器设计中伺服电机及驱动器的选型》。

施耐德 SoMachine 控制器应用及编程指南

李幼涵　编著



机　械　工　业　出　版　社

本书围绕施耐德 SoMachine 控制平台的运动控制器、逻辑控制器、变频控制器和人机界面控制器的硬件应用环境，讲解了基于 IEC61131 标准的 SoMachine 软件的编程规则。使设计人员对机器控制的设计和编程不局限在一种编程格式，也不拘泥于只对逻辑状态进行编程。他们可以根据工艺要求而采用顺序流程图（SFC）方式规划结构，采用结构文本（ST）方式进行复杂工艺运算和调节计算，采用梯形图（LD）方式处理各种逻辑和工艺过程，采用功能图（FBD）方式进行同一功能的反复使用和对通信功能的搭建。同时书中对组成机器控制的常用功能，例如编码器高速计数、总线通信、数据交换及运动控制功能的编辑做了阐述。对在程序调试过程中用到的虚拟人机操作界面，变量的曲线记录等应用也做了描述。

本书适合自动控制、机器制造设计工程师阅读，也适合用作相关专业学生和研究生的教学用书。

图书在版编目（CIP）数据

施耐德 SoMachine 控制器应用及编程指南/李幼涵编著. —北京：机械工业出版社，2014.4
ISBN 978-7-111-46531-7

Ⅰ.①施… Ⅱ.①李… Ⅲ.①可编程序控制器-指南 Ⅳ.①TP332.3-62

中国版本图书馆 CIP 数据核字（2014）第 082827 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）
策划编辑：林春泉　责任编辑：赵　任
责任印制：刘　岚　责任校对：刘秀丽　程俊巧
北京京丰印刷厂印刷
2014 年 6 月第 1 版·第 1 次印刷
184mm×260mm·24.5 印张·599 千字
0 001—4 000 册
标准书号：ISBN 978-7-111-46531-7
　　　　　ISBN 978-7-89405-375-6（光盘）
定价：69.00 元（含 1CD）

凡购本书,如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
社服务中心：（010）88361066	教材网：http://www.cmpedu.com
销售一部：（010）68326294	机工官网：http://www.cmpbook.com
销售二部：（010）88379649	机工官博：http://weibo.com/cmp1952
读者购书热线：（010）88379203	**封面无防伪标均为盗版**

序

进入21世纪后，中国经济尤其是机械制造业的高速发展，为各种自动化技术的应用提供了巨大的应用舞台。一方面基于总线控制、运动控制以及对工艺进行复杂算法的高速运算等自动化技术，极大地提升了机器的生产效率和控制精度与运行速度；另一方面多变的市场需求，需要机器的设计更加柔性和灵活，机器的上市时间更加缩短，甚至机器的全生命周期成本更加优化。

施耐德电气作为国际知名的全球能效管理专家的代表，世界上第一台可编程序控制器（Programmable Logic Controller，PLC）的发明者，于近年推出了全新系列、专门针对OEM市场的中小型PLC。不仅如此，施耐德电气还同步推出了基于CoDeSys的统一软件控制平台SoMachine，集成了对PLC、伺服、变频器和人机界面等施耐德电气自动化硬件的统一控制。SoMachine软件平台支持多种编程控制语言无缝衔接，对总线通信技术及复杂运动控制技术提供了强大的支持，也为客户设计出更加人性化的交互界面以及更简便的操作提供了便利。

李幼涵老师是施耐德电气资深的自动化专家，也是施耐德电气官方认定的公司全球顶尖技术专家之一。他在机器自动化领域经营多年，曾多次到国外接受各种先进技术的培训。目前，他已经将施耐德电气SoMachine控制器的学习心得和应用经验编写成书，一是向国内OEM企业技术人员、大专院校师生和业界学者、专家等进行全面的展示，二是为他们提供一种全新的技术设计、开发思路、应用经验，以解决实际应用中的问题。

施耐德电气真诚地愿为中国机械制造工业的不断提升、技术的不断进步、创新不断涌现，向市场推出接受度更高的产品。本书必定会成为机械制造业中重要的自动化技术参考资料和培训教材。期待着本书早日面世以飨读者。

施耐德电气（中国）有限公司 工业事业部

OEM业务副总裁　庞邢健

2014年3月

前　　言

自从2004年接触并应用CoDeSys以来，一直想把这种编程方法和使用经验写出来。尤其是在机器制造行业，各种围绕以运动控制为核心、以传动为核心的解决方案层出不穷。对符合IEC61131标准编程和PLCopen及SoftMotion的需要越来越多。设计人员对机器控制的设计和编程已经不局限在一种编程格式，也不拘泥于只对逻辑状态进行编程。他们需要根据工艺要求而采用顺序流程图（SFC）的方式规划结构，采用结构文本（ST）的方式进行复杂工艺运算和调节计算，采用梯形图（LD）的方式处理各种逻辑和工艺过程，采用功能图（FBD）的方式进行同一功能的反复使用和对通信功能的搭建。总之，目前对机器设计的要求已经不只是几台电动机的转动和接触器、继电器的吸合。它的解决方案已上升到机器的柔性控制。包括传动电动机的运转，伺服电动机的电子齿轮运动和电子凸轮的运动，机械手臂的快速抓取以及机器人控制等复杂运动。对相关自动控制元件的组合，涉及方方面面通过各种总线连接的远程I/O、人机界面、执行器和开关按钮到远程的以太网通信。对于这么复杂的一种编程方法，要把它和具体的机器设计结合起来，给工程师带来实用的参考价值和利益，确实需要大量的应用案例积累和不断的实践体验来感受和强化这种编程方法带来的效益并把它简洁地表达出来，使工程师们易于接受，这确实不是一件容易的工作。好在我是做技术支持的，可以直接面对各地的技术工程师和客户，了解他们的需求。我还是做培训工作的，了解不同行业的使用者提出和遇到的问题。并且我有机会在德国和法国进行过长时间的研修。这些工作经历和同事们、客户们不断的支持，使我有条件把这些体验和经验收集起来，经过分析、整理，编写成书，呈现给大家。目前，随着施耐德电气进军机器制造解决方案领域，推出了SoMachine平台，使编程有了一个硬件平台的支撑，让大家在实际应用中感受到这种编程方法带来的高效和柔性。他山之石，可以攻玉。愿大家掌握这种编程方法，使各种机器的设计不断创新，为企业带来效益。

由于涉及的知识、概念较多，为了便于学习，本书分为上、下两册。上册为基础篇，主要介绍了一些控制器硬件和软件系统。对软件的6种编程语言SFC、CFC、FBD、ST、LD和IL进行了论述和给出了应用案例；对控制系统经常用到的现场总线通信，如MODBUS，CANopen的组态，对编程进行了详细阐述并详细地讲解应用案例；对程序的调试，仿真和执行过程的监测，记录以及可视界面的应用也做了详细的阐述，同时还介绍了当今流行的如何利用公共资源，诸如智能手机等设备进行无线操控和监视的实现方法。下册为高级篇，主要讨论了运动控制的复杂应用，涉及电子凸轮和CNC的应用设计，诸如电子凸轮和CNC的曲线设计，曲线在计算机和控制器之间的传输；电子凸轮点和CNC位置点的变量设计和修改；CNC G代码文件的编辑生成和传输；滑台机械手和水平关节机械手的动作设计等。由于本人水平有限，编辑组织的不尽完美，请各方面专家指正并提出宝贵意见，以便进一步修改。

本书适用于机器设计、运动控制设计的工程师，特别是数控 CNC，机器人设计的技术人员和研究人员以及相关专业大学生和研究生进行 PLCopen 和 Softmotion 的教学参考书。

施耐德电气（中国）有限公司 工业事业部
高级工程师　全球技术专家　李幼涵
2014 年 3 月

目　　录

序
前言
第 1 章　绪言 …… 1
1.1　概论 …… 1
1.2　当前的机器控制设计 …… 1
1.3　学习目标 …… 1
1.4　本书的使用 …… 2
第 2 章　控制器硬件平台 …… 3
2.1　硬件概览 …… 3
2.2　经济型 M218 逻辑控制器的结构 …… 3
2.3　优化型 M238 逻辑控制器的结构 …… 4
2.4　高性能 M258 逻辑控制器的结构 …… 6
2.5　优化型 LMC058 运动控制器的结构 …… 9
2.6　优化型 IMC 传动控制器的结构 …… 12
2.7　XBTGC 人机界面控制器 …… 14
2.8　优化型 TM2 扩展模块 …… 14
2.9　高性能 TM5 扩展模块 …… 15
第 3 章　控制器软件编程平台 SoMachine …… 17
3.1　软件概览 …… 17
3.2　SoMachine 编程软件简介 …… 17
3.3　计算机系统要求 …… 19
3.4　SoMachine 编程软件界面 …… 19
3.5　SoMachine 编程软件首页 …… 20
3.6　一般功能菜单 …… 23
3.7　项目操作流程 …… 23
3.8　属性页面 …… 26
第 4 章　项目的管理和建立 …… 29
4.1　概览 …… 29
4.2　使用空项目启动 …… 29
4.3　以现有项目启动 …… 35
4.4　以模板项目启动 …… 38
4.5　扩展模板的添加 …… 41
第 5 章　编辑控制器程序 …… 45
5.1　概览 …… 45
5.2　POU 程序创建 …… 51
5.3　任务配置 …… 57
5.4　PLC 的程序仿真 …… 59
5.5　CoDeSys 编程语言 …… 61
5.5.1　POU ST 结构文本编程方式 …… 62
5.5.2　POU IL 指令表编程方式 …… 70
5.5.3　POU LD 梯形图编程方式 …… 75
5.5.4　POU FBD 功能块图编程方式 …… 80
5.5.5　POU CFC 连续功能图编程方式 …… 85
5.5.6　POU SFC 顺序功能图编程方式 …… 90
5.6　功能块的建立 …… 103
第 6 章　数据类型 …… 106
6.1　支持的数据类型，命名规则 …… 106
6.1.1　支持的数据类型概览 …… 106
6.1.2　变量命名规则 …… 106
6.2　局域变量（Local）和全局变量（Global） …… 111
6.3　数据单元类型（Data Unit Type） …… 115
6.3.1　数组变量（Array） …… 115
6.3.2　结构变量 …… 116
6.3.3　枚举变量（Enumeration） …… 127
6.4　带有物理地址的变量 …… 131
第 7 章　在线配置和组态任务 …… 133
7.1　网关组态 …… 133
7.2　应用操作 …… 134
7.3　PLC 设置和管理 …… 141
7.4　内置 I/O 的组态 …… 144
7.5　任务组态和运行 …… 147
第 8 章　程序的调试和诊断 …… 155
8.1　状态栏 …… 155
8.2　信息窗 …… 156
8.3　日志 …… 158
8.4　调试工具 …… 159
8.4.1　仿真 …… 159
8.4.2　变量监测和强制 …… 160
8.4.3　设置断点（Breakpoint） …… 161

8.4.4　曲线记录 …… 174
8.5　系统变量和功能 …… 186
第9章　可视界面的创建及应用 …… 191
9.1　SoMachine 编程软件集成的可视界面 …… 191
9.2　视图的建立及编辑 …… 192
9.3　复用可视界面 …… 233
第10章　应用库文件 …… 236
10.1　应用库文件概述 …… 236
10.2　库文件管理 …… 238
10.3　库文件的建立 …… 241
第11章　Modbus 通信 …… 250
11.1　Modbus 通信介绍 …… 250
11.2　通信的组态 …… 251
11.3　总线扫描模式 IOScanner 组态 …… 253
11.4　直接请求模式 …… 262
第12章　CANopen 通信 …… 271
12.1　CANopen 总线基础 …… 271
12.2　过程数据对象和服务数据对象通信模式 …… 278
12.3　CANopen 总线的组态 …… 279
12.4　直接请求的数据交换 SDO …… 292
第13章　总线设备工具及设备类型管理器应用 …… 296
13.1　FDT 及 DTM 概述 …… 297
13.2　DTM 的组态 …… 298
第14章　简单运动控制功能的实现 …… 312
14.1　采用脉冲控制的设计方法 …… 312
14.2　采用 CANopen 总线控制的设计方法 …… 322
14.3　采用 CANmotion 总线控制的设计方法 …… 330
第15章　运动控制器中编码器的应用 …… 335
15.1　编码器的硬件接线 …… 335
15.2　LMC058 中编码器的应用 …… 337
第16章　基于以太网的数据交换 …… 346
16.1　以太网组态 …… 346
16.2　以太网服务 FTP 文本传输协议 …… 348
16.3　SoMachine 协议 …… 353
16.4　客户/服务器模式 …… 358
16.5　连接无处不在 …… 360
第17章　高速计数器 …… 364
17.1　高速计数器简介 …… 364
17.2　计数模式 …… 368
17.3　组态 …… 372
17.4　简单和复杂计数的功能块 …… 374
17.5　PTO-脉冲串输出 …… 377
17.6　案例 …… 378

第 1 章　绪　　言

1.1　概论

本书是一本详细论述和指导有关机器控制方案设计的工具书，它论述了符合 IEC61131-3 编程标准的 SoMachine 控制开发系统的使用。它涉及了 PLCOPEN，SOFTMOTION 这些通用的编程功能块，使得读者在设计机器的逻辑控制、通信控制、运动控制和 CNC 功能时更加通用开放，易于掌握。本书中，同时介绍了施耐德机器控制的硬件平台，这个平台包括了以逻辑控制为核心的 PLC，以运动控制为核心的运动控制器，以传动为核心的传动控制器 IMC 和以人机界面为核心的人机界面控制器。这些控制器的编程均采用名为 SoMachine 的控制开发软件，这就使得这种软件的使用不是曲高和寡，而是脚踏实地针对机器制造商提出的各种要求，实实在在地完成了各种工艺过程，执行各种动作并记录了动作的轨迹。在软、硬件的搭配下，你可以体验、练习这些灵活便捷的编程环境和指令，以及易于修改和诊断的程序模式。读者可以把它当作一本教科书，用于教学和练习；也可以把它当成工具书，随时查阅有关指令和案例，对设计者有很好的参考作用。

1.2　当前的机器控制设计

当前的机器控制设计，加入的控制元素越来越多，从传感器、开关按钮到变频器、伺服电动机、人机界面、PLC。要把这些控制元素组合起来，并且使得机器的控制变得高效、稳定和安全，则要求控制器不仅有良好的逻辑编辑能力，还要有运动控制、传动、人机界面的编程能力；不仅有适合逻辑编程的梯形图（LD）模式，还要有适合复杂运算的结构文本（ST）模式；不仅有结构清晰，功能显见的控制功能图编程（CFC）模式，还要有步序清晰、功能规划合理、调试方便的顺序功能图（SFC）模式；不仅有单机的控制能力，还要有良好的扩展和与周边设备的通信能力。这种由简洁的控制结构组合的开发平台，使得机器控制的设计更加柔性，更加适应机器的各种工作要求，更加快速响应市场的需求。

1.3　学习目标

学完本书，你应该达到的目标是：

➢ 应用 SoMachine 建立一个控制器程序。

➢ 使用 SoMachine 的各种编程语言，建立程序组织块 POU 及应用程序。

➢ 组态程序组织单元以不同的任务模式来执行。

➢ 编辑简单运动控制程序来控制伺服驱动器 Lexium 05 或 Lexium 32A。

➢ 建立和使用用户程序库。

➢ 建立和使用用户数据类型。
➢ 使用和建立调试、检测工具。
➢ 使用 Modbus 与远程 I/O 模块 OTB 通信。
➢ 采用 CANopen 与远程设备通信。
➢ 理解和使用高速输入的各种计数模式。
➢ 组态和使用内置高速输出。
➢ 建立人机界面 XBTGT 和 M238/M258 PLC 的通信。

1.4 本书的使用

本书针对的读者是应用 CoDeSys 开发环境的新用户，以 SoMachine 为基础的机器控制设计工程师和施耐德电气公司的应用工程师。本书分为很多章节，读者可以通过模仿、练习书中练习题和案例，逐步搭建自己的系统，熟悉各种指令和变量的应用。书中的章节特意将软件、硬件分出单独的章节，读者可以单独学习软件，练习各种编程。要想实施一个机器的设计时，可以查相关硬件的章节，做出可实施的系统。

第 2 章　控制器硬件平台

2.1　硬件概览

在不同的机器工艺要求下，有些控制偏重简单的逻辑控制，有些机器偏重传动的调速控制，有些机器的工艺要求多个传动设备同步或精确定位，有些设备想简化到只通过一台人机界面就可控制机器的运转。对于各种各样的要求，催生了适应各种要求的柔性控制系统 SoMachine，如图 2-1 所示。在这个系统中包含了 4 种硬件控制器。它们是以逻辑控制为核心的逻辑控制器，以变频传动为核心的传动控制器，以运动控制为核心的运动控制器和以人机界面为核心的人机界面控制器。这些控制器都采用同一个软件 SoMachine 来编程，在一个平台下，就实现了对不同机器的控制。这种柔性的组合，使得机器的设计周期大大缩短，快速响应了市场的需求。

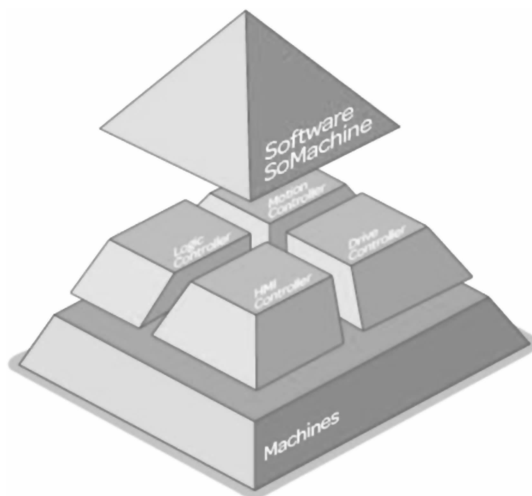


图 2-1　一台机器的 4 种硬件平台

2.2　经济型 M218 逻辑控制器的结构

在一些简单型机器上，机器的工艺动作较多地使用逻辑控制，附加一些对高速计数输入和高速脉冲输出的要求以及对通信的一般需求。对这种控制器的要求就是控制结构简捷、经济实惠。在 SoMachine 平台，M218 逻辑控制器就基本满足了经济实惠、功能齐全的要求。基本硬件结构如图 2-2 所示。

M218 逻辑控制器具有的基本结构是：

1）具有数字量的输入和输出，其中数字量的输入可以设置为高速计数器（HSC），接收 2 路 A/B 相的脉冲输入或是 4 路单相的脉冲输入，输入频率可以达到 100kHz。数字量的输出也可以设置为 2 路脉冲串（PTO）输出，最大输出频率可达 100kHz。可以控制两台步进电动机或伺服电动机。用于对步进电动机或伺服电动机的速度控制和位置控制。

2）具有模拟量的输入和输出。

3）具有两个串行口，可以设置为 Modbus 的主站控制从站设备，也可以设置为从站接受主站设备的控制。串口 1 还可以设置为 SoMachine 网络连接在 SoMachine 网络内，像网络邻居那样实现各个设备的网络互连。

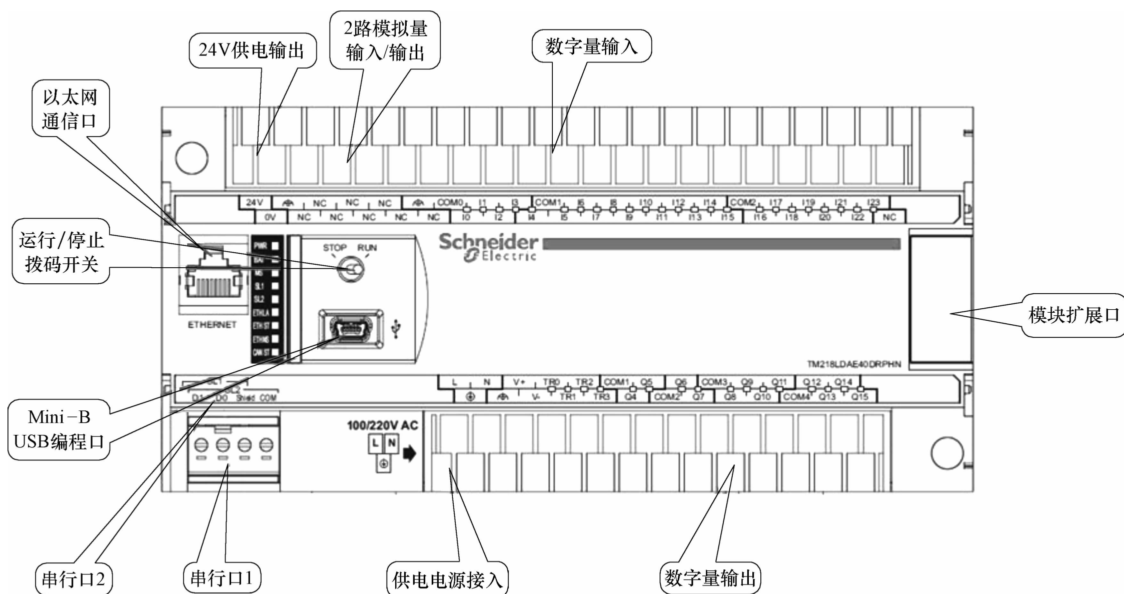


图 2-2　M218 逻辑控制器基本硬件结构

4）具有以太网通信口，支持 Modbus TCP SERVER；Modbus TCP CLIENT。

5）具有 MINI USB 编程口，用于程序的编制和调试。

6）具有模块扩展接口，可以扩展到 264 点。

在配置时，打开 SoMachine 软件，把合适的 M218 逻辑控制器拖入配置框内，就可以开始一台机器的控制设计了。M218 逻辑控制器的配置如图 2-3 所示。



图 2-3　M218 逻辑控制器的配置

2.3　优化型 M238 逻辑控制器的结构

在有些机器的控制中，需要通信更快，接入的高速输入口更多，和专用的功能块支持更

多，由此产生了对优化型逻辑控制器的需求，M238 逻辑控制器就是满足上述需求的一种控制器。其基本硬件结构如图 2-4 所示。

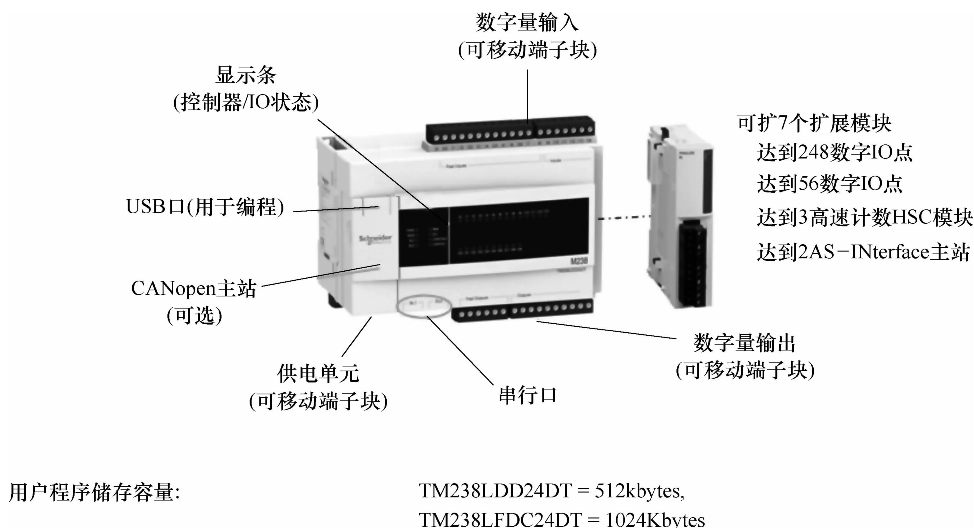


图 2-4 M238 逻辑控制器基本硬件结构

M238 逻辑控制器具有的基本结构是：

1）具有数字量的输入和输出，其中数字量的输入可以设置为高速计数器（HSC），接收 2 路 A/B 相的脉冲输入或是 8 路单相的脉冲输入，输入频率可以达到 100kHz。数字量的输出也可以设置为 2 路脉冲串（PTO）输出或脉宽调制（PWM），最大输出频率可达 100kHz。可以控制两台步进电动机或伺服电动机。用于对步进电动机或伺服电动机的速度控制和位置控制。高速计数（HSC）和脉冲输出（PTO）/脉宽调制（PWM）如图 2-5 所示。

2）具有两个串行口，可以设置为 Modbus 的主站控制从站设备，也可以设置为从站接受主站设备的控制。还可以设置为 Modbus IO SCANNER，这样配置好从站设备的参数或控制地址，就可以方便地在输入、输出字上填写参数和控制命令来控制从站设备或修改从站设备的参数。串口还可以设置为 SoMachine 网络连接在 SoMachine 网络内，像网络邻居那样实现各个设备的网络互连。串口 1 也可以设置为 RS-232 通信。

3）具有 CANopen 总线主站通信口，连接 16 台从站设备，通信速度达到 1M。M238 逻辑控制器的 CANopen 总线结构如图 2-6 所示。

4）具有 MINI USB 编程口，用于程序的编制和调试。

5）具有模块扩展接口，可以扩展到 248 点。

在配置时，打开 SoMachine 软件，把合适的 M238 逻辑控制器拖入配置框内，就可以开始一台机器的控制设计了。M238 逻辑控制器的配置如图 2-7 所示。

要注意的是 M238 逻辑控制器型号中有些是带有 S0 的，在带有 S0 的控制器中，提供了针对包装、起重和输送行业中典型的控制功能块，供用户调用。如包装行业的堆垛、张力、温度控制，起重行业的防摇、纠偏和同步控制等。



图 2-5　高速计数（HSC）和脉冲输出（PTO）/脉宽调制（PWM）



图 2-6　M238 逻辑控制器的 CANopen 总线结构

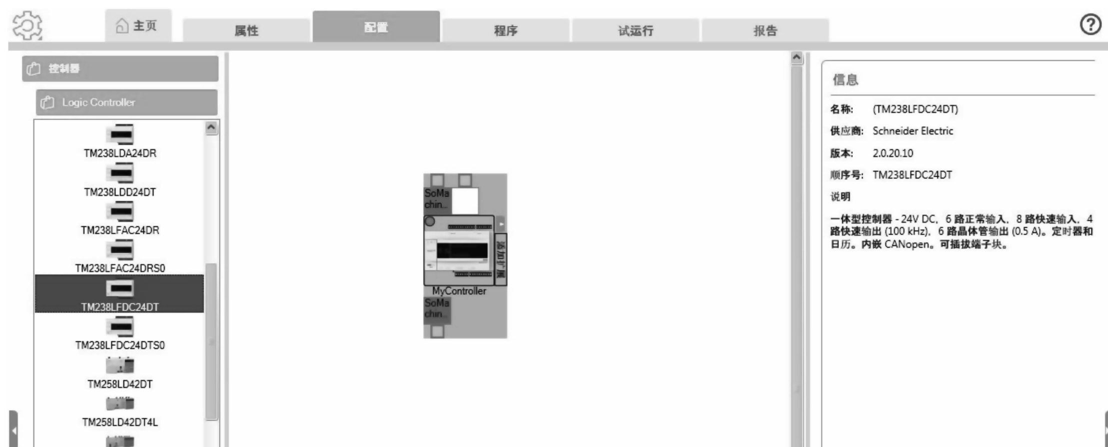


图 2-7　M238 逻辑控制器的配置

2.4　高性能 M258 逻辑控制器的结构

在有些机器的控制中，不仅需要通信更快，还需要接入的高速输入口更多，专用的功能块支持更多，程序的运行更快，挂接在总线上的设备更多，和控制点数更多。由此产生了对高性能逻辑控制器的需求，M258 逻辑控制器就是满足上述需求的一种控制器。高性能逻辑控制器 M258 结构如图 2-8 所示。

M258 逻辑控制器具有的基本结构是：

1）具有数字量的输入和输出，其中数字量的输入可以设置为高速计数器（HSC），接收 2 路 A/B 相的脉冲输入或是 8 路单相的脉冲输入，输入频率可以达到 200kHz。数字量的输出也可以设置为 2 路脉宽调制（PWM）输出，最大输出频率可达 100kHz。可以控制两台

步进电动机或伺服电动机的速度。也可以改变脉宽来控制加热。高速脉冲的输入和输出如图 2-9 所示。

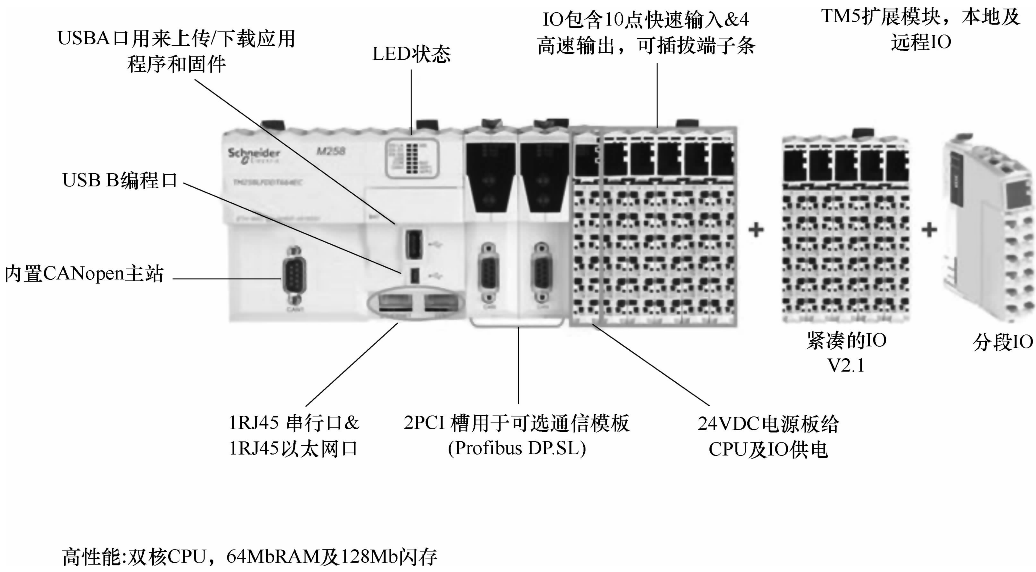


图 2-8　高性能 M258 逻辑控制器结构

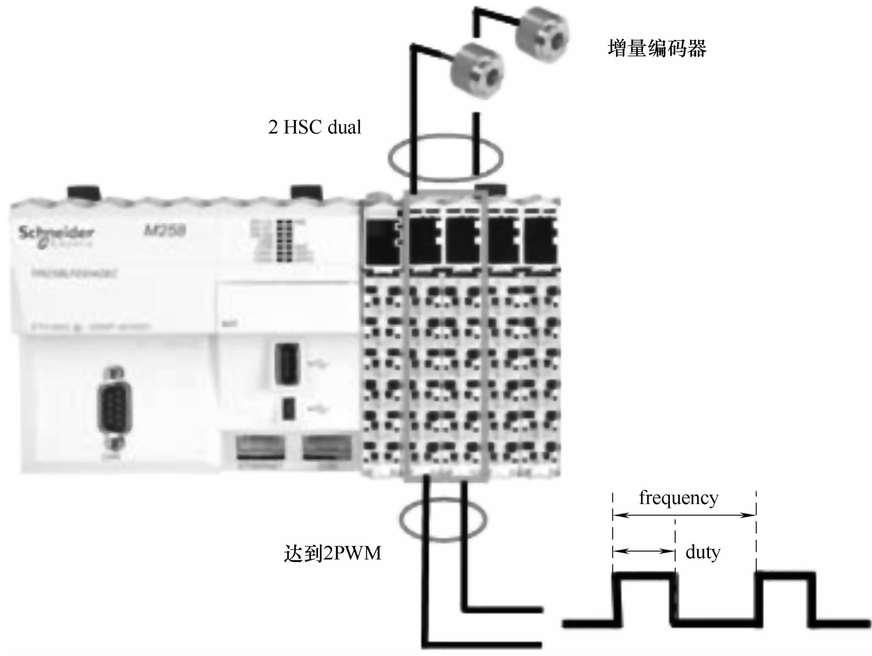


图 2-9　高速脉冲的输入和输出

2）具有 1 个串行口，可以设置为 Modbus 的主站控制从站设备，也可以设置为从站接受主站设备的控制。还可以设置为 Modbus IO SCANNER，这样配置好从站设备的参数或控制地址，就可以方便地在输入、输出字上填写参数和控制命令来控制从站设备或修改从站设备的参数。串口还可以设置为 SoMachine 网络连接在 SoMachine 网络内，像网络邻居那样实

现各个设备的网络互连，如和人机界面连接交换数据。也可以加上 MODEM。串口的连接如图 2-10 所示。

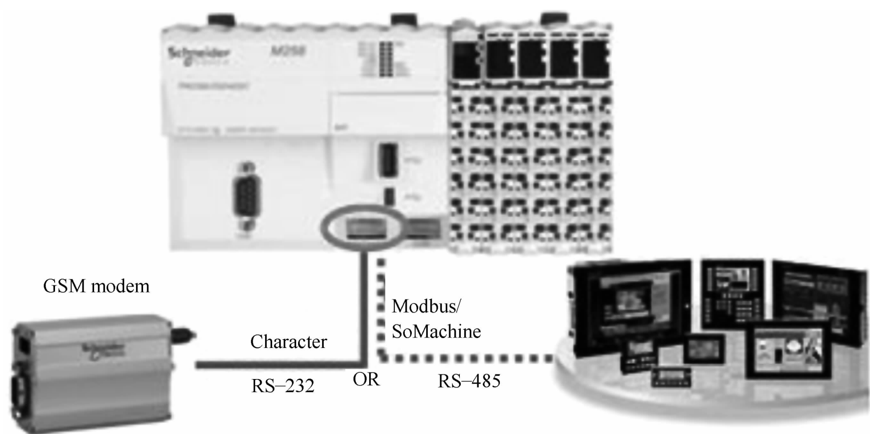


图 2-10　M258 逻辑控制器串口的连接

3）具有 CANopen 总线主站通信口，可以连接 32 台从站设备，通信速度达到 1Mbit/s。M258 逻辑控制器 CANopen 总线的连接如图 2-11 所示。

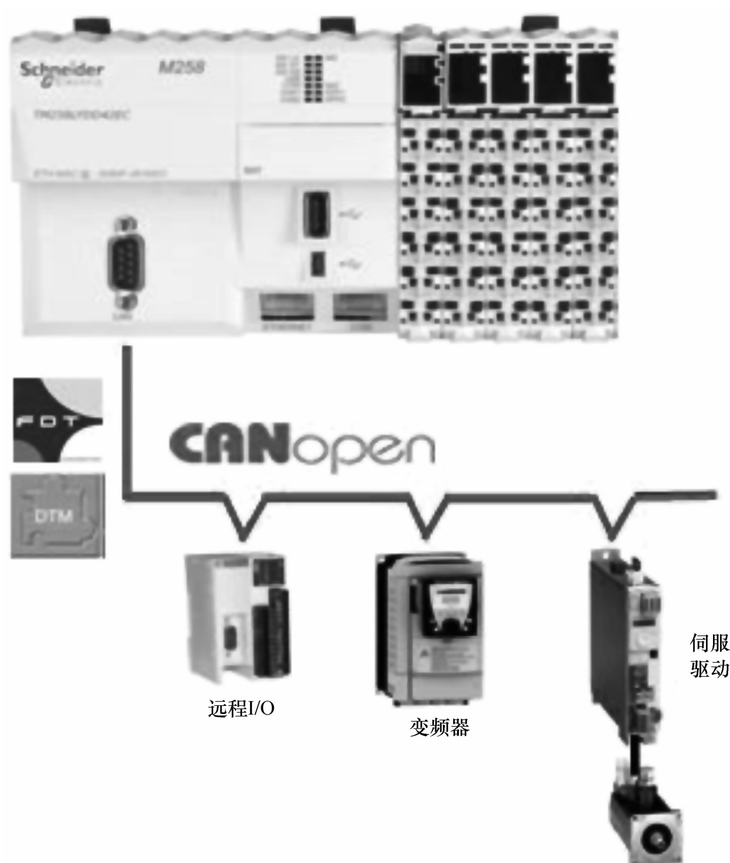


图 2-11　M258 逻辑控制器 CANopen 总线的连接

4）具有 MINI USB 编程口，用于程序的编制和调试。还有一个 USB 口用于连接 USB 存储卡上传和下载程序、数据及刷新固件。

5）具有一个以太网口，支持以太网通信，如图 2-12 所示可以通过以太网编程调试，也可以通过以太网连接人机界面和多个 M258 逻辑控制器的互连。

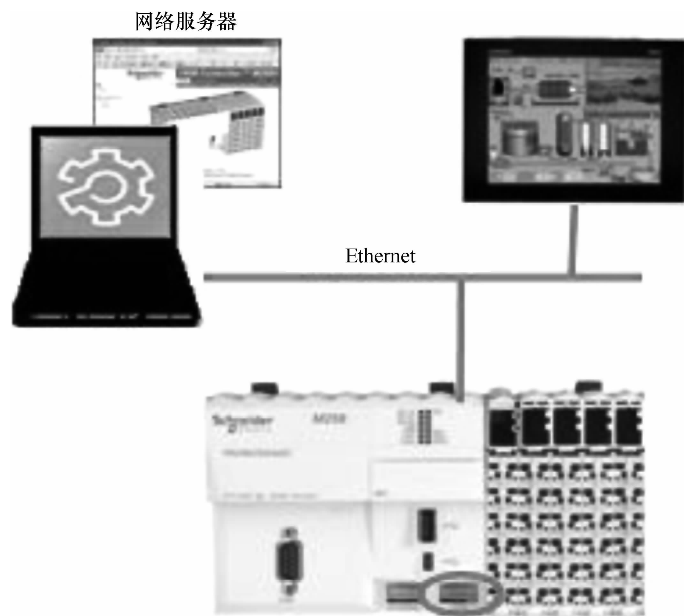


图 2-12　M258 逻辑控制器以太网的连接

6）具有模块扩展接口，可以扩展到 2400 点。由于是背板扩展，所以数据的传输速度为 12Mbit/s。这里需要注意的是扩展的模块 TM5 有两种形式：一种模块是紧凑型的模块，如 TM5C24D18T（24 入，18 出）；另一种模块是片式的模块，如 TM5SDI12D。

同样要注意的是 M258 逻辑控制器型号中有些是带有 S0 的，在带有 S0 的控制器中，提供了针对包装、起重和输送行业中典型的控制功能块，供用户调用。如包装行业的堆垛、张力、温度控制，起重行业的防摇、纠偏和同步控制等。

在配置时，打开 SoMachine 软件，把合适的 M258 逻辑控制器拖入配置框内，就可以开始一台机器的控制设计了。M258 逻辑控制器的配置如图 2-13 所示。

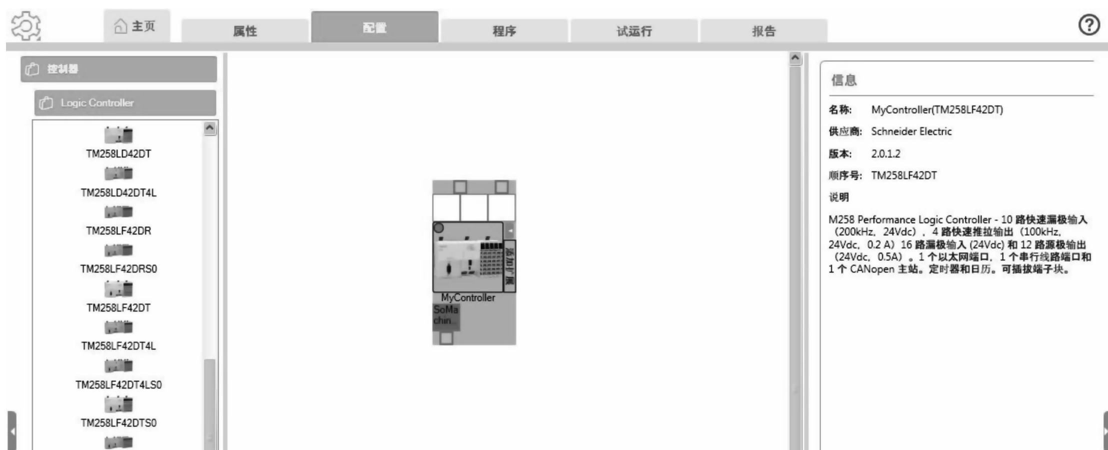


图 2-13　M258 逻辑控制器的配置

2.5　优化型 LMC058 运动控制器的结构

在有些机器的控制中，不仅需要逻辑控制，还需要复杂的定位控制和多轴的同步控制。

例如，多轴的电子齿轮同步，主从轴的电子凸轮曲线控制，主从轴的相位同步控制以及运行曲线的插补控制等。这就要求整个控制系统具有以运动控制为核心的控制结构。LMC058 运动控制器就是以运动控制为核心的控制器。LMC058 运动控制器的结构如图 2-14 所示。

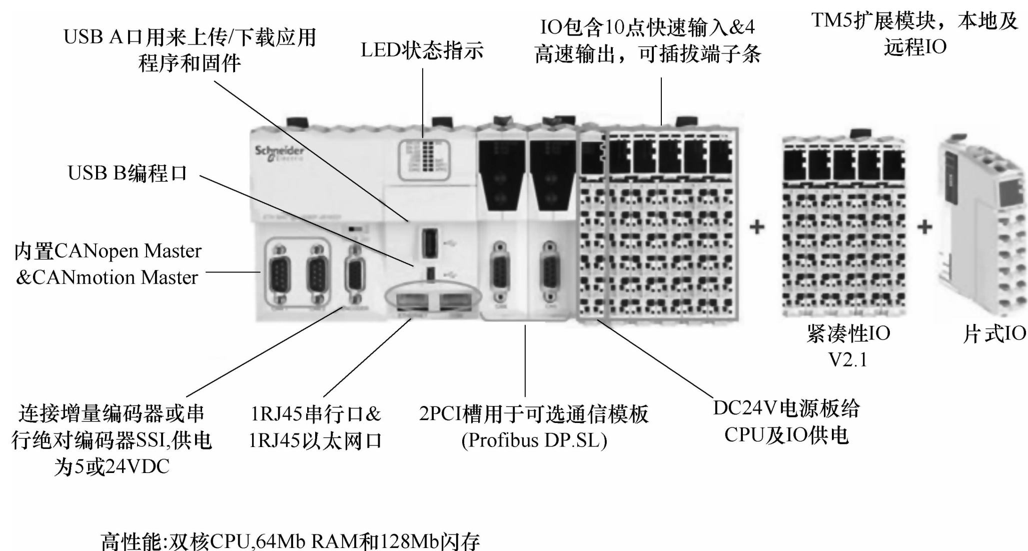


图 2-14　LMC058 运动控制器的结构

LMC058 运动控制器具有的基本结构是：

1）具有数字量的输入和输出，其中数字量的输入可以设置为高速计数器（HSC），接收 2 路 A/B 相的脉冲输入或是 8 路单相的脉冲输入，输入频率可以达到 200kHz。数字量的输出也可以设置为 2 路脉宽调制（PWM）输出，最大输出频率可达 100kHz。可以控制两台步进电动机或伺服电动机的速度。也可以改变脉宽来控制加热。高速脉冲的输入和输出如图 2-15 所示。

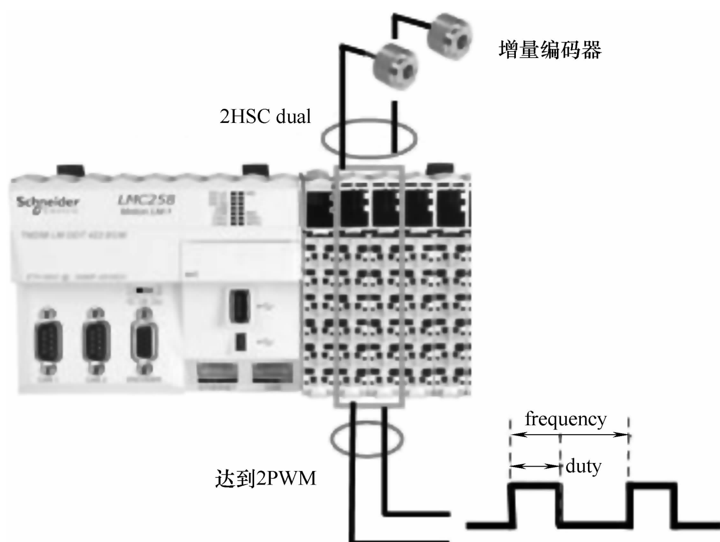


图 2-15　高速脉冲的输入和输出

2）具有 1 个串行口，可以设置为 Modbus 的主站控制从站设备，也可以设置为从站接受主站设备的控制。还可以设置为 Modbus IO SCANNER，这样配置好从站设备的参数或控制地址，就可以方便地在输入、输出字上填写参数和控制命令来控制从站设备或修改从站设备的参数。串口还可以设置为 SoMachine 网络连接在 SoMachine 网络内，像网络邻居那样实现各个设备的网络互连，如和人机界面连接交换数据。也可以加上 MODEM。LMC058 运动控制器串口的连接如图 2-16 所示，连接数据终端设备（DTE）、打印机、条码扫描，数据通信设备（DCE）、调制解调器和转换器。

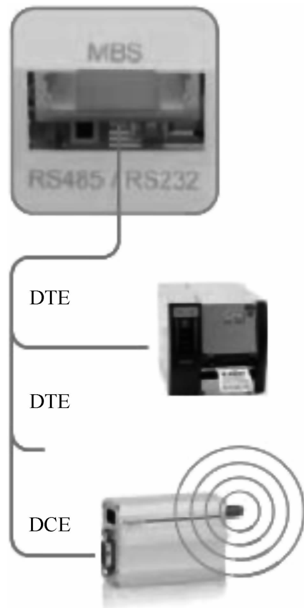


图 2-16　LMC058 运动控制器串口的连接

3）具有 CANopen 总线主站通信口，可以连接 32 台从站设备，通信速度达到 1Mbit/s。

LMC058 运动控制器 CANopen 总线的连接如图 2-17 所示。

具有 CANmotion 总线主站通信口，可以带 8 轴同步，8 轴同步时间为 4ms。可以做 2 维线性及圆弧插补，可以做电子齿轮和电子凸轮。还具有主轴编码器输入，可以做实轴和虚轴，CANmotion 和主轴编码器输入如图 2-18 所示。

4）具有 MINI USB 编程口，用于程序的编制和调试。还有一个 USB 口用于连接 USB 存储卡上传和下载程序、数据及刷新固件。

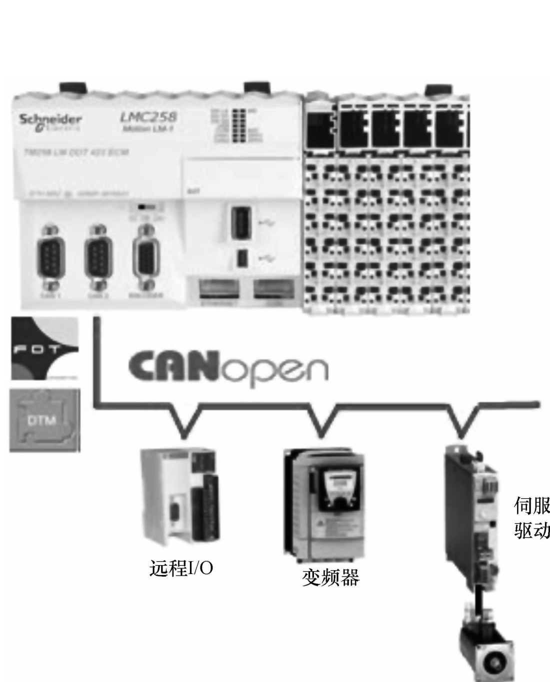


图 2-17　LMC058 运动控制器 CANopen 总线的连接

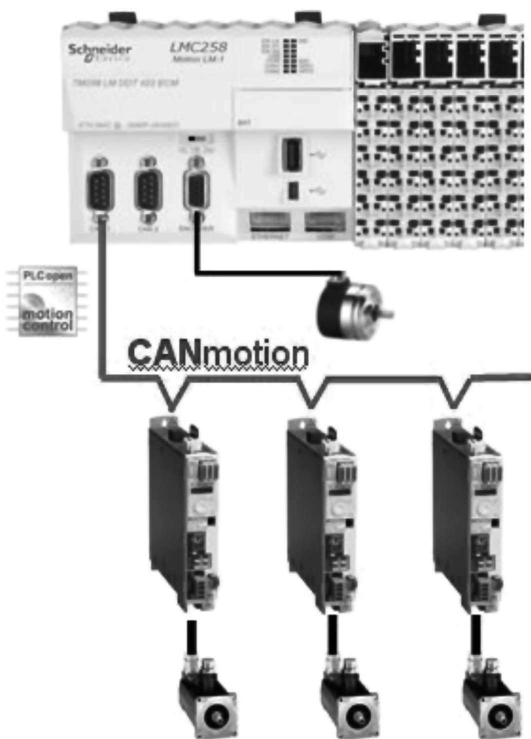


图 2-18　CANmotion 和主轴编码器输入

5）具有一个以太网口，支持以太网通信，如图 2-19 所示可以通过以太网编程调试，也可以通过以太网连接人机界面和多个 PLC 的互连。

6）具有模块扩展接口，可以扩展到 2400 点。由于是背板扩展，所以数据的传输速度为 12Mbit/s。这里需要注意的是扩展的模块 TM5 有两种形式：一种模块是紧凑型的模块，如 TM5C24D18T（24 入，18 出）；另一种模块是片式的模块，如 TM5SDI12D。

同样要注意的是 LMC058 逻辑运动控制器型号中也有些是带有 S0 的，这些带 S0 的控制器中，提供了针对包装、起重和输送行业中典型的控制功能块，供用户调用。如包装行业的堆垛、张力、温度控制、飞剪控制和辊剪控制等。

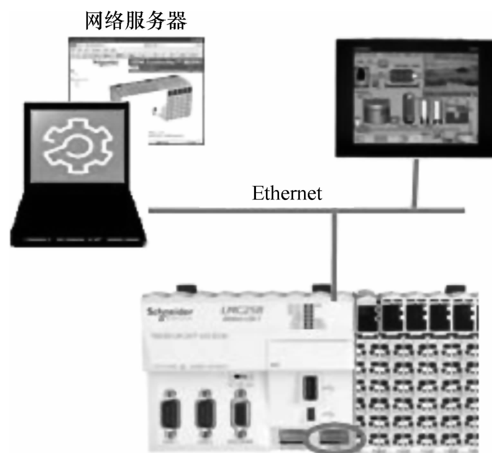


图 2-19　LMC058 逻辑运动控制器以太网的连接

在配置时，打开 SoMachine 软件，在 Motion Controller 栏目，把合适的 LMC058 逻辑运动控制器拖入配置框内，就可以开始一台机器的控制设计了，LMC058 逻辑运动控制器的配置如图 2-20。

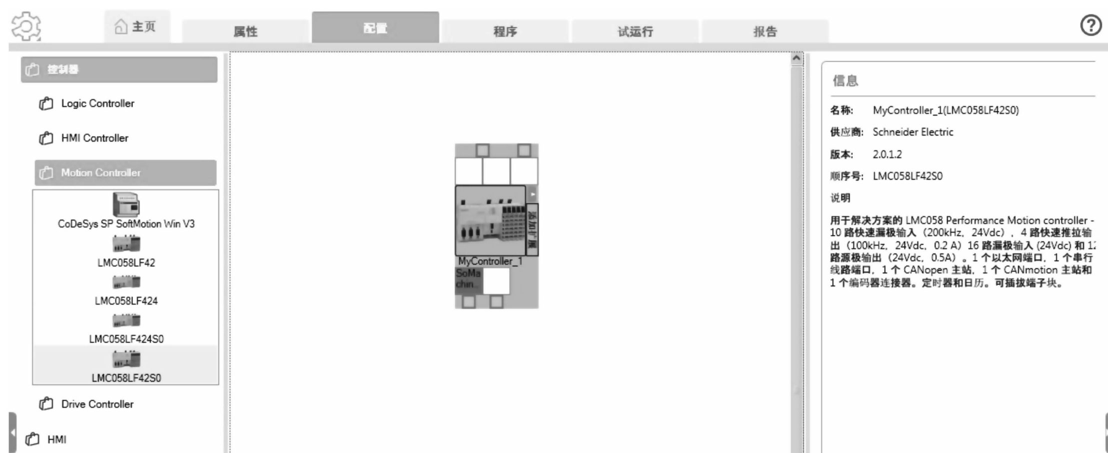


图 2-20　LMC058 逻辑运动控制器的配置

2.6　优化型 IMC 传动控制器的结构

有些机器设备的控制是以变频调速为主的，例如塔机、港口机械等，这些机器在起吊和搬运物体时，不仅要求快速，而且还要求准确、安全。相应的控制就要具有防摇、纠偏、同步起吊等功能。配置在变频器上的 IMC 传动控制器嵌入控制单元使变频器的应用具有了智能化。它不仅具有控制、通信功能，还无缝利用了本体变频的一切资源，延伸了变频的功能。它是由一块智能控制卡嵌在变频器上组成的，如图 2-21 所示。

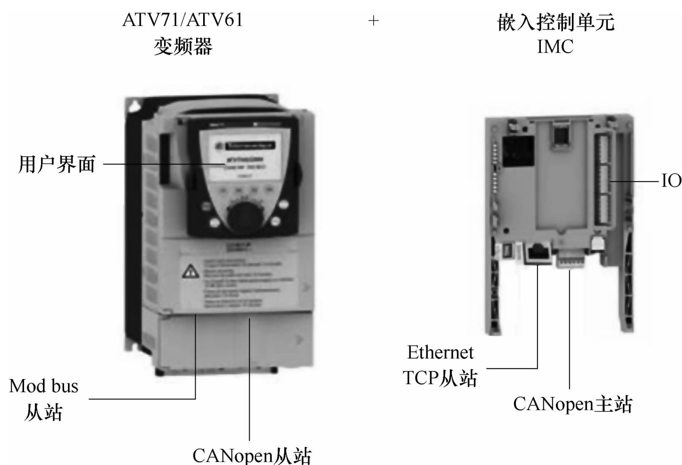


图 2-21　IMC 传动控制器和变频的组合

IMC 传动控制器的基本结构是：

1）具有数字量的输入和输出，其中有 6 个高速输入点，这些点的输入可以设置为高速计数器（HSC），接收 2 路 A/B 相的脉冲输入或是 2 路单相的脉冲输入，输入频率可以达到 100kHz。具有 2 路模拟量输入和 2 路模拟量的输出。

2）具有 MINI USB 编程口，用于程序的编制和调试。

3）具有一个以太网口，支持以太网通信，可以通过以太网编程调试，也可以通过以太网连接人机界面和多个 PLC 的互连。

4）具有 CANopen 总线主站通信口，可以连接 32 台从站设备，通信速度达到 1Mbit/s。

IMC 传动控制器的控制结构如图 2-22 所示。

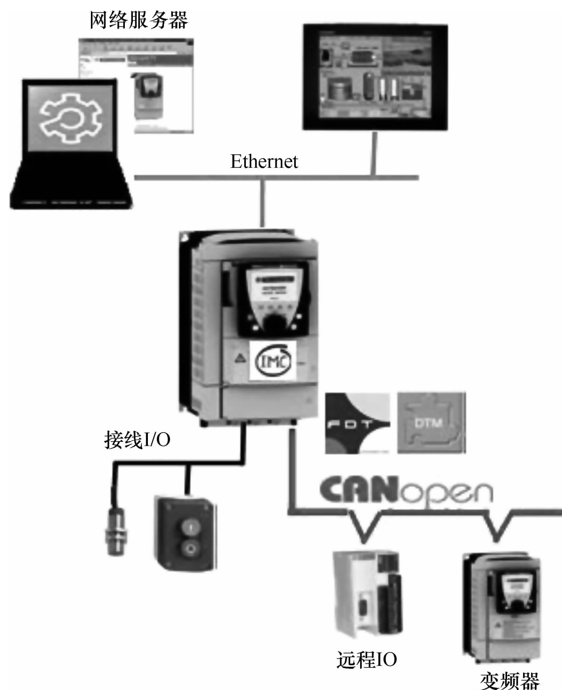


图 2-22　IMC 传动控制器的控制结构

在配置时，打开 SoMachine 软件，在 Drive Controller 栏目，把 ATV-IMC 拖入配置框内，就可以开始一台机器的控制设计了，如图 2-23 所示。

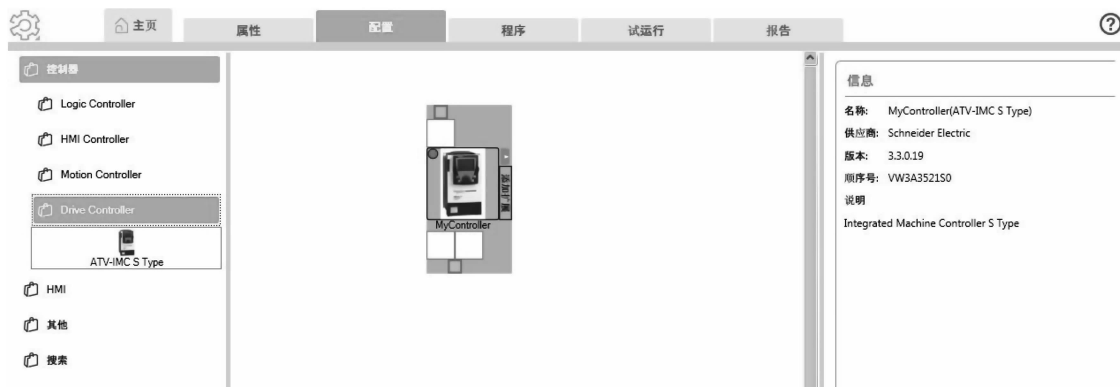


图 2-23　配置一个 ATV-IMC

2.7　XBTGC 人机界面控制器

当机器需要一个人机界面就可以控制的简单结构时，我们可以选择以人机界面为核心的控制方式，如图 2-24 所示。



图 2-24　人机界面为核心的控制结构

2.8　优化型 TM2 扩展模块

TM2 扩展模块是为 M218 逻辑控制器和 M238 逻辑控制器提供的扩展模板，在 SoMachine 配置界面，点击 M218 逻辑控制器添加扩展，就会出现添加设备的列表，如图 2-25 所示。

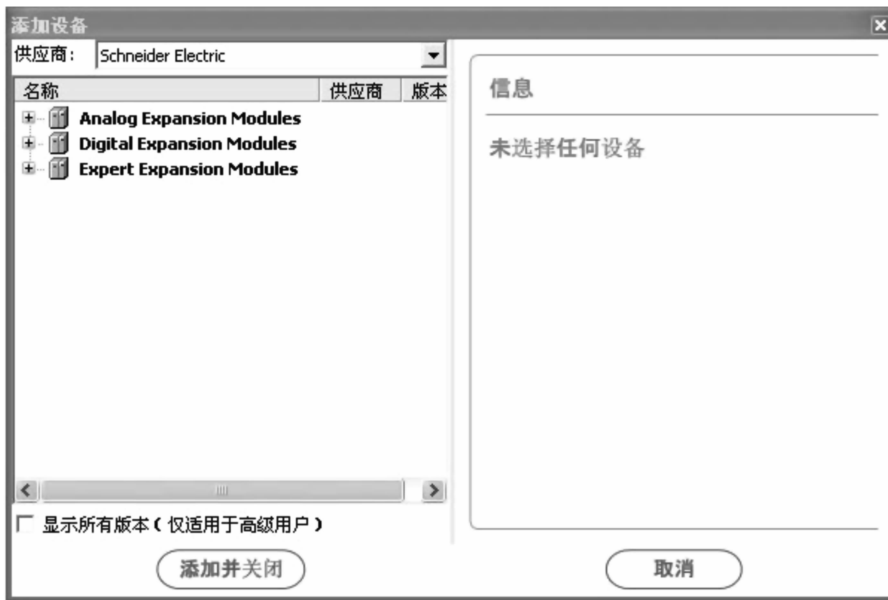


图 2-25　添加设备列表

选择合适的扩展，点击添加并关闭，就完成了一次添加。

2.9　高性能 TM5 扩展模块

TM5 扩展模块是为 M258 逻辑控制器和 LMC058 运动控制器提供的扩展模板，在 SoMachine 配置界面，点击 M258 逻辑控制器或 LMC058 运动控制器添加扩展，就会出现添加设备的列表，如图 2-26 所示。

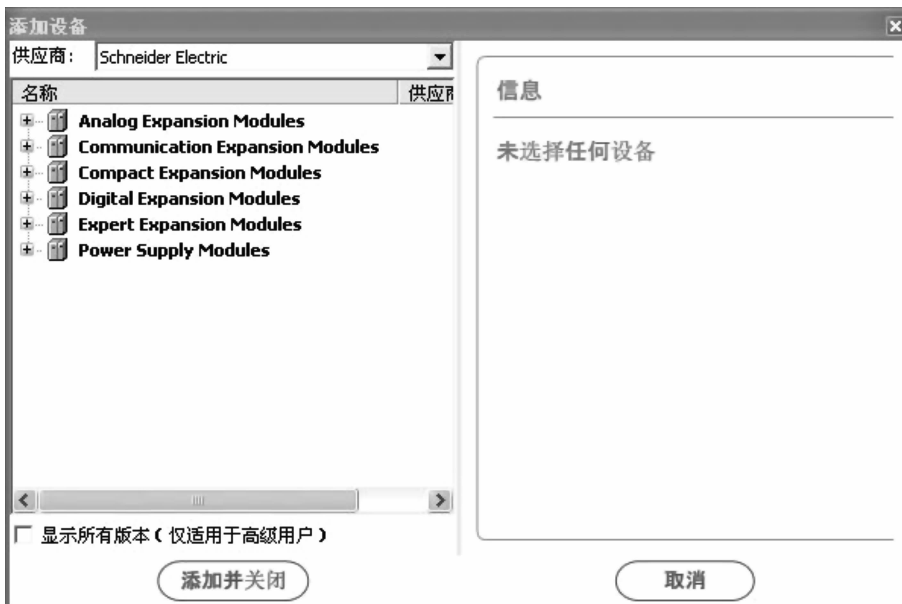


图 2-26　TM5 扩展模块列表

选择合适的扩展，点击添加并关闭，就完成了一次添加。当添加的模板过多时，编译会给出报警，提示供电不足，需要加入一块电源模块，这时一定要在供电不足的扩展模块前面加入电源模块，也就是说电源模块不能放在最后。另外，有的工程师在使用 SoMachine V2 版时，添加扩展，会找不到 Compact Expansion Modules，这需要运行一个补丁程序 io modules extended catalog setup.exe，然后打开 SoMachine 重新建立新的项目即可。补丁程序可以从施耐德公司网站下载。

第 3 章　控制器软件编程平台 SoMachine

3.1　软件概览

SoMachine 是一个专业的、高效的，针对于机器制造业开放的解决方案软件，它集开发、组态和调试为一体，包含了逻辑控制、电动机控制、人机界面和相应的网络通信功能。我们之所以说它是一个解决方案软件，是因为在逻辑控制器、传动控制器、运动控制器和人机界面这 4 个硬件平台下的所有编程都是由一个软件即 SoMachine 完成的。在整个机器的调试过程中，用一根编程电缆，可以完成所有控制器的编程以及人机界面的组态编程。下载时也可以完成对所有控制器的程序一次下载。当连接多个控制器时，也不需要分别连接各自的编程口来编程，只要从一个编程口就可以对多个连在网络上的控制器进行编程和调试。这样就大大方便了编程操作，减少了连接次数。由于覆盖 4 种硬件平台，就使得控制功能大大加强，可以适应从简单到复杂的各种机型，达到了控制系统的柔性化，提高了对市场需求的响应速度。针对包装、起重和输送，我们还开发了专用功能块，使得编程速度大大提高。例如包装的飞剪控制、辊剪控制和张力控制等。

3.2　SoMachine 编程软件简介

SoMachine 编程软件是基于 CoDeSys（Controller Development System） V3 开发而成的，所以它具有 CoDeSys 的一般功能和通用应用模式。它支持的功能如下。

1. 标准编程语言

SoMachine 作为标准的编程语言，它包括了 6 种符合 IEC61131-3（International Electrotechnical Commission）的编程语言。根据不同的要求，应用程序可以采用不同的语言来混合编制。从而极大地方便各种解决方案。这些编程语言是：

☐功能块图（FBD）；

☐顺序功能流程图（SFC）；

☐结构文本（ST）；

☐指令表（IL）；

☐梯形图（LD）；

☐连续功能流程图（CFC）。

2. 控制器编程支持

☐用户建立的功能块（FBs）；

☐用户建立的功能；

☐数据单元类型（DUTs）；

☐在线修改；

☐观察窗；
☐变量曲线监视（曲线记录）；
☐断点，分步执行；
☐仿真；
☐程序和机器运行的可视及操作界面。

3. 基于人机界面的支持

☐图形库包含多于 4000 个 2D 和 3D 元素；
☐简单的制图工具（点、线、矩形、椭圆等）；
☐可重置元素（按钮、开关、柱状图等）；
☐配方（32 组，256 个配方包含最大 1024 种元素）；
☐执行表；
☐报警；
☐打印；
☐Java 脚本；
☐多媒体文件支持格式：wav、png、jpg、emf、bmp；
☐趋势变量。

4. 基于对运动控制的支持

☐内置设备的组态和调试；
☐电子凸轮 CAM 曲线编辑器；
☐变量运行曲线记录；
☐变频、伺服和步进驱动的功能块库；
☐可视监视和操作界面。

5. 全局支持

☐用户访问和浏览；
☐项目文档打印；
☐项目比较（控制）；
☐基于关联机器和已有机器的变量共享；
☐程序库版本管理。

6. 内置现场总线组态

☐主站：
- CANopen；
- CANmotion；
- Modbus 串行口；
- AS-interface。

☐可接入总线：
- Profibus-DP；
- Ethernet IP；
- Modbus TCP。

7. 应用程序库

□标准库：

– PLCopen 用于运动控制的程序库。

□行业解决方案库：

– 包装功能库；

– 输送功能库；

– 起重功能库。

3.3　计算机系统要求

由于软件比较大，而且可以使用中文和英文，所以对计算机的配置有一些要求。计算机硬件的配置见表 3-1。

表 3-1　计算机硬件的配置

描　　述	最 低 配 置	推 荐 配 置
处理器	Pentium V, 1.8GHz, Pentium M, 1.0 GHz 或相应	Pentium V, 3.0GHz, Pentium M, 1.5GHz 或相应
RAM	1024MB	2048MB
富余的硬盘空间	3.5G	4G
光驱	DVD	DVD
显示	分辨率：1024 × 786	分辨率：1280 × 1024
外设	USB 接口	USB 接口
网络	以太网口	以太网口

软件要求安装下列系统：

Windows XP Professional 或 Windows Vista32 位。

3.4　SoMachine 编程软件界面

启动 SoMachine 编程软件，在 Windows 启动界面，沿以下路径选择 SoMachine：Start » Programs » Schneider Electric » SoMachine> » SoMachine 或在桌面点击图标 

就可启动 SoMachine 编程软件，如图 3-1 所示。



图 3-1 SoMachine 编程软件主页

3.5 SoMachine 编程软件首页

1. 可视图形用户界面

导引 SoMachine 编程软件使用的是直观和闪亮的视觉图示。这个可视图形界面（Graphical User Interface，GUI）可以方便地为你的设计项目搭建不同的优化结构平台。用户在这个界面上查看结构是否完整，在设计项目时添加要增加的执行任务。在这个优化的工作区里，只看到适合当前任务的设备和相关信息，没有任何多余和无用的信息。

2. 主页选择界面

成功启动 SoMachine 编程软件以后，可看到如图 3-2 所示的主页图形界面，通过这个界面，可以进入到各个功能界面。

3. 主页功能

在主页左侧任务选择条上，我们可以执行列出的 SoMachine 编程软件任务，如图 3-3 所示。

在显示现有机器栏目下，可以点击浏览现有项目，打开存储项目文件的文件夹，找出已经存储的项目文档。也可以提取存档的打包压缩文件，来解压缩并打开这个文档。

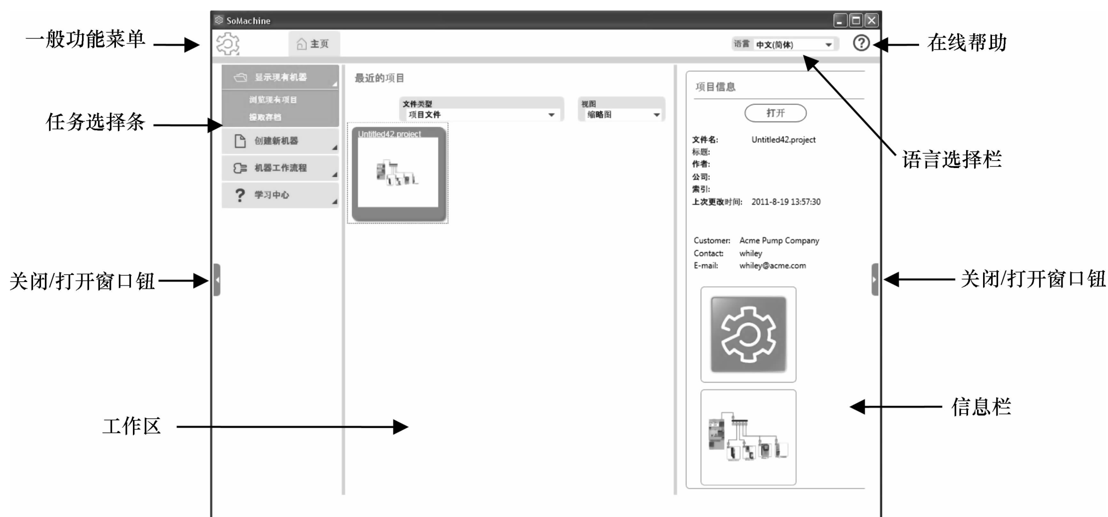


图 3-2　SoMachine 编程软件主页图形界面



图 3-3　主页任务选择

点击创建新机器，得到图 3-4 所示的在创建新机器栏目下的功能。

在创建新机器栏目下，可以点击使用空项目启动来建立一个新的项目；也可以点击使用标准项目启动来选择一个控制模板作为项目；也可以点击使用 TVD 架构启动，从已做好测试，在可用的架构中选择合适的平台来构建项目；也可以点击使用应用程序启动从输送、起重、包装的应用程序中开始建立项目；也可以点击使用现有项目启动，来选择继续一个项目。

点击机器工作流程，如图 3-5 所示。在机器工作流程栏目下，点击试运行机器 - 使用项目启动，即打开存储文档的文件夹，找出要启动的文件并打开进入调试。

点击试运行机器-从设备上载项目，即进入连接硬件平台并从此平台上把程序上传到 PC 进入程序调试。

点击更新固件后，开始对 M238 逻辑控制器的固件进行更新操作。



图 3-4　创建新机器



图 3-5　机器工作流程

点击学习中心，进入学习中心栏目，如图 3-6 所示。在这个栏目下，点击快速概览，如图 3-6 所示。可以查看一个动态短篇，来概览 SoMachine 编程软件的特点和优势，以及相关编程概念。



图 3-6　学习中心功能

点击培训手册，会看到培训 SoMachine 编程软件的一般知识和概念手册，通过这个手册，可以大概了解 SoMachine 编程软件的软件编程概念。

点击 E-Learning 会得到一个演示幻灯片，播放 SoMachine 编程软件的知识和相关控制平台的介绍。点击示例，会得到一些成功案例的说明程序，以便在编程时作为参考。总之，在

空余时间，通过学习中心提供的这几种方式，可以对 SoMachine 编程软件有一个大概的了解和体验，并就遇到的问题查看示例，以获得帮助。

3.6　一般功能菜单

一般功能菜单如图 3-7 所示。



图 3-7　一般功能菜单

点击图 3-7 所示一般功能菜单的图标后，会打开下拉的窗口。在这个菜单中，打开首页或对首选项的文件夹路径进行设定，也可以打开帮助文件或查看许可和注册情况，也可以退出 SoMachine。在编辑当前文件和编程时，点击一般功能菜单图标，就可以对文件进行保存或指定路径命名文件夹的保存或对文档打包压缩进行保存及发送。

3.7　项目操作流程

优化项目开发

SoMachine 编程软件用户界面是直观和图形化的界面，可以按照引导图标来完成整个项目的流程，而不用花费多余的时间。

项目启动以后，就会要求你填写相关项目信息，以便确定项目，如图 3-8 所示。

定义了项目范围以后，就可以建立或设计包括控制器在内的硬件控制结构了。借助 SoMachine 编程软件图形化设计，可以很方便和直观地建立起硬件结构，包括扩展模块及扩展控制器，人机界面及现场总线通信，如图 3-9 所示。



图 3-8　项目信息

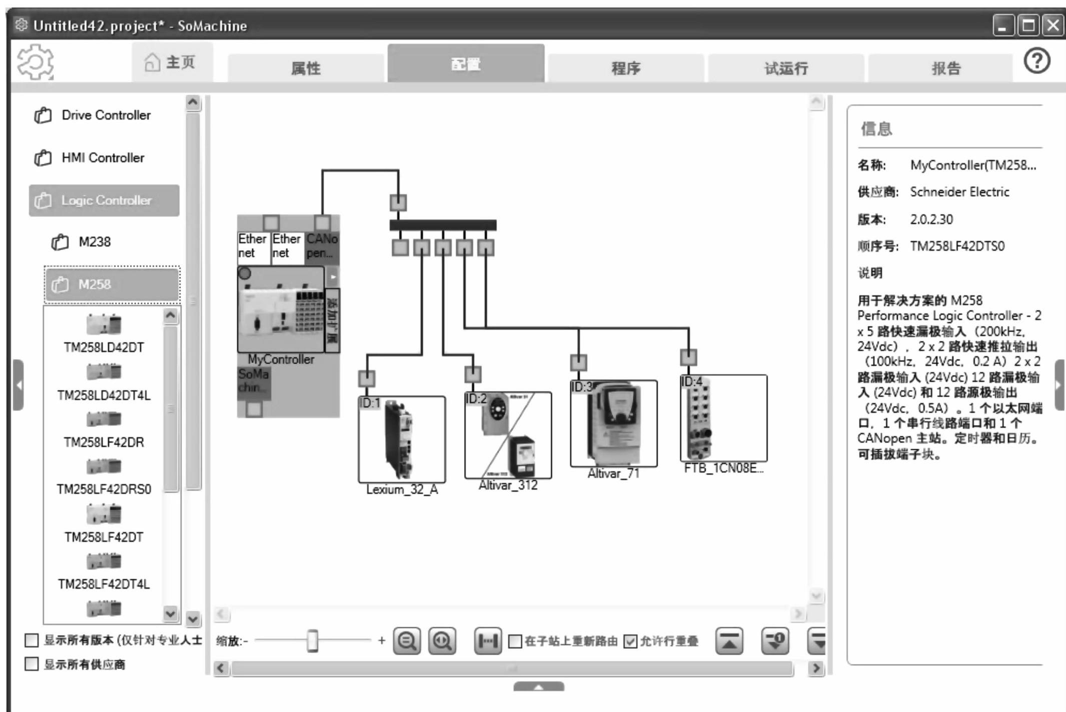


图 3-9　图形化硬件结构设计

点击程序图标，可打开项目的编程平台进行应用程序的开发。几乎所有 CoDeSys 功能在这个平台都是有效的，如图 3-10 所示，可以应用 6 种语言编写机器的控制程序。

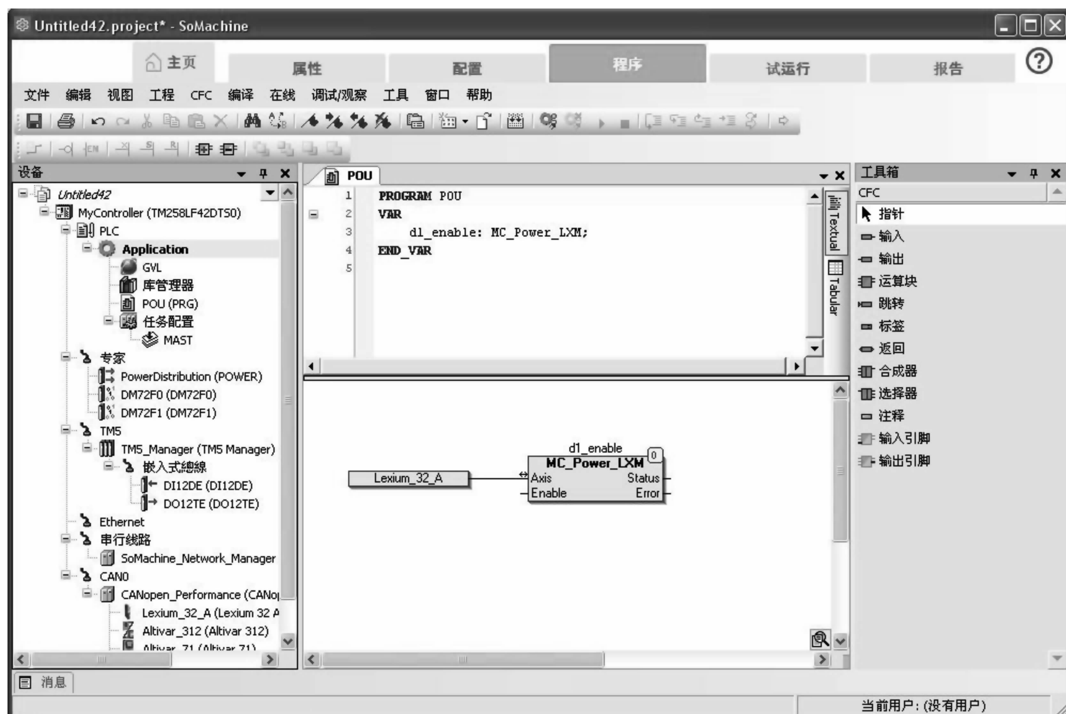


图 3-10　应用程序开发平台

进入调试阶段，可以在线进入控制器及连接到的设备上，查看设备状态。也可以下载应用程序或刷新控制器固件，如图 3-11 所示。

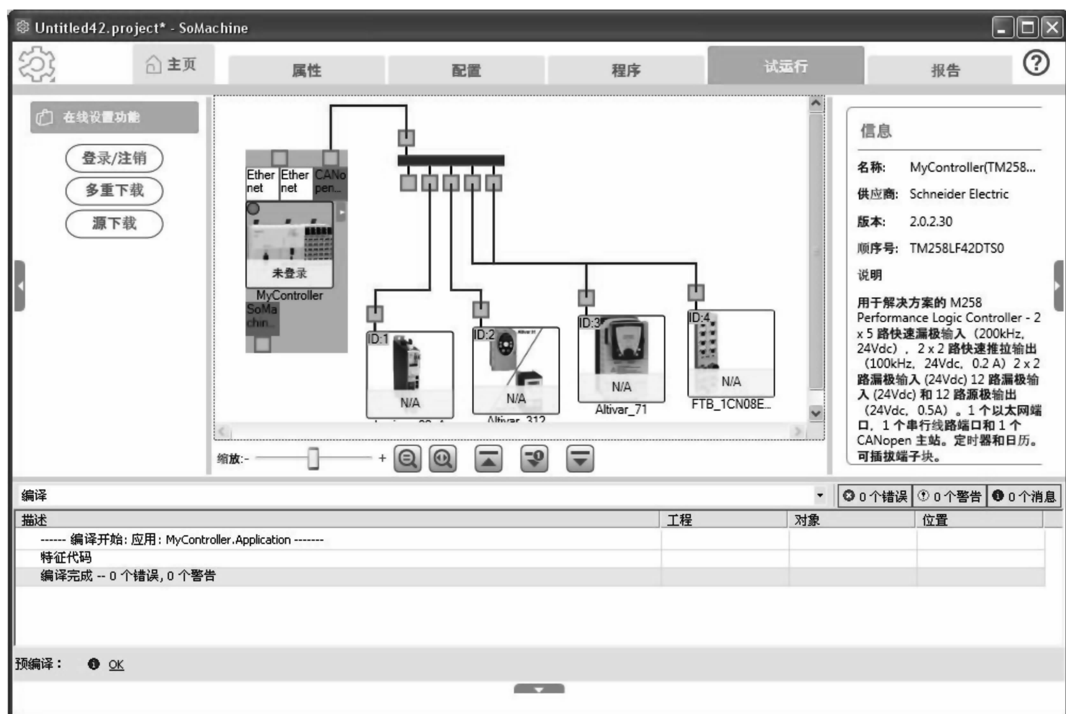


图 3-11　调试状态

完成整个项目后，就可以自动形成文档报告。这个报告包括硬件配置、软件编程和项目信息，非常便于归档和交流。项目存档如图 3-12 所示。



图 3-12　项目存档

3.8　属性页面

1. 添加项目信息

属性页面允许用户添加项目信息。项目的文字和图形都可以添加。由于这是在工作区内可看到的项目摘要，所以它会帮助你快速找到每个所需的项目，而不用把整个项目都打开。这个页面只有当一个项目已经完成时才可查询显示。属性页面的建立如图 3-13 所示。



图 3-13　属性页面的建立

属性页面提供如下功能：

☐一般信息；

☐描述；

☐客户信息。

2. 一般信息

在属性页面下点击常规，可以添加文件信息。如文件名、存储路径。也可以填写设计者信息。

3. 描述文档

点击说明，打开界面如图 3-14 所示。在这个界面下，可以导入机器或设备照片。同时在配置视图里显示你为这台机器控制配置好的控制结构图。



图 3-14　客户信息和配置界面

4. 客户信息文档

在自定义信息处，可以输入客户的相关信息和客户资料作为附件，如图 3-15 所示。定义好的信息在信息框内打勾，在项目信息栏目的显示中就会出现。这样不用打开自定义信息，就可看到客户信息的关键字段。



图 3-15　客户资料和信息

第 4 章　项目的管理和建立

4.1　概览

前面介绍了 SoMachine 编程软件使用的一个流程。现在我们根据客户的需求建立并设计一个项目。首先运行 SoMachine 编程软件，并在首页选择显示现有机器或创建新机器。在显示现有机器栏目下，可以浏览过去建立的项目打开并继续你的项目或修改或完善这个项目；也可以提取存档并打开一个压缩项目文件进行查看、参考或编辑。在创新机器栏目下，建立一个新的机器项目。为了使你的设计尽快进入实施，可以选择合适的行业、应用的成熟的应用设计模板启动，也可以选择和你的控制结构相近的、成熟的 TVD 架构启动，这些做好的结构和程序模板可以使你的编程工作事半功倍。当编程遇到问题时，打开这些现成的模板，可以参考所编程序，看看问题出在哪里；也可以参考模板的控制结构和编程，看看这个结构是如何实现的；当然，你也可以平地起高楼，选择一个空项目进行启动，完成一个从无到有的任务实施。

4.2　使用空项目启动

打开主页，如图 4-1 所示，点击使用空项目启动。

弹出界面，对项目命名一个文件名，然后点击“Save”，把启动文件存储在设定的文件夹内，如图 4-2 所示。

这个存储的文件，以后可以在显示现有机器栏目下调用。

存储这个文件后，进入到属性页面，可以先填写项目的有关信息，如图 4-3 所示。

填写完项目信息，就可以进入配置阶段，如图 4-4 所示。在这个阶段中，可以根据客户的要求，选择硬件平台，然后把相应的控制器拖动到工作区内。

例如，我们选择逻辑控制器平台的控制器 TM238LFDC24DT，把此控制器拖到工作区内，如图 4-5 所示。

把控制器拖入工作区后，可以建立相应的外围设备。如通过 CANopen 总线控制一台变频器，一台伺服驱动器。这样，就可以点击工作区内控制器的 CAN 总线框，来添加设备。



图 4-1　创建新机器

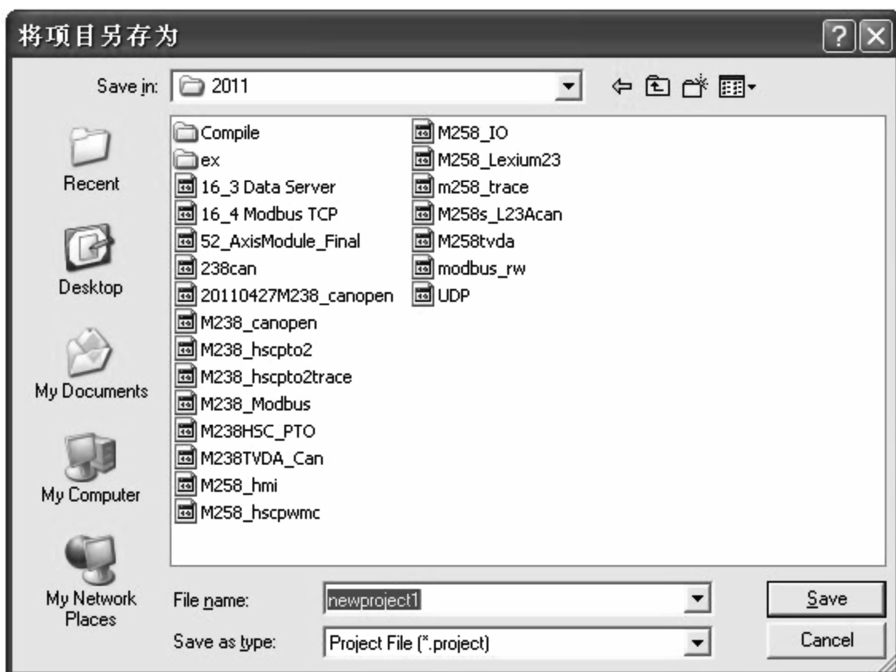


图 4-2　建立文件



图 4-3　填写项目信息



图 4-4　配置硬件控制结构

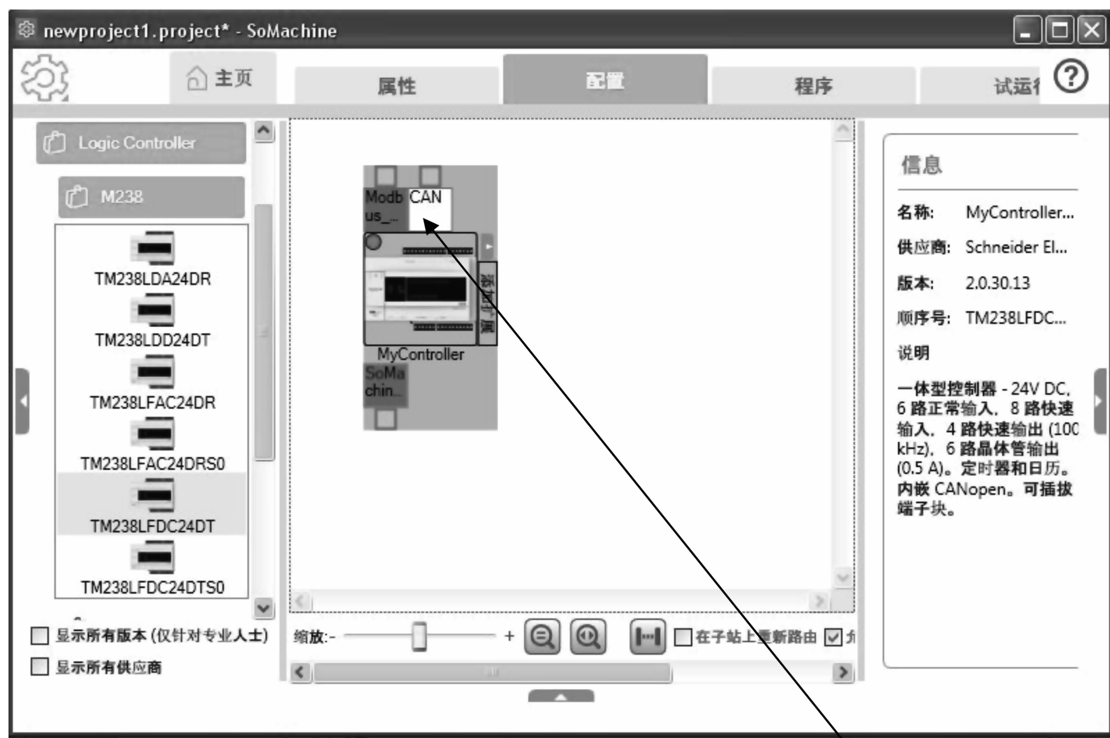


图 4-5　把控制器拖到工作区

点击总线框后，会弹出图 4-6 所示界面，选择 CANopen Optimized，然后点击“添加并关闭”。



图 4-6　添加总线口

这样，我们加上了一个 CANopen 口，如图 4-7 所示。

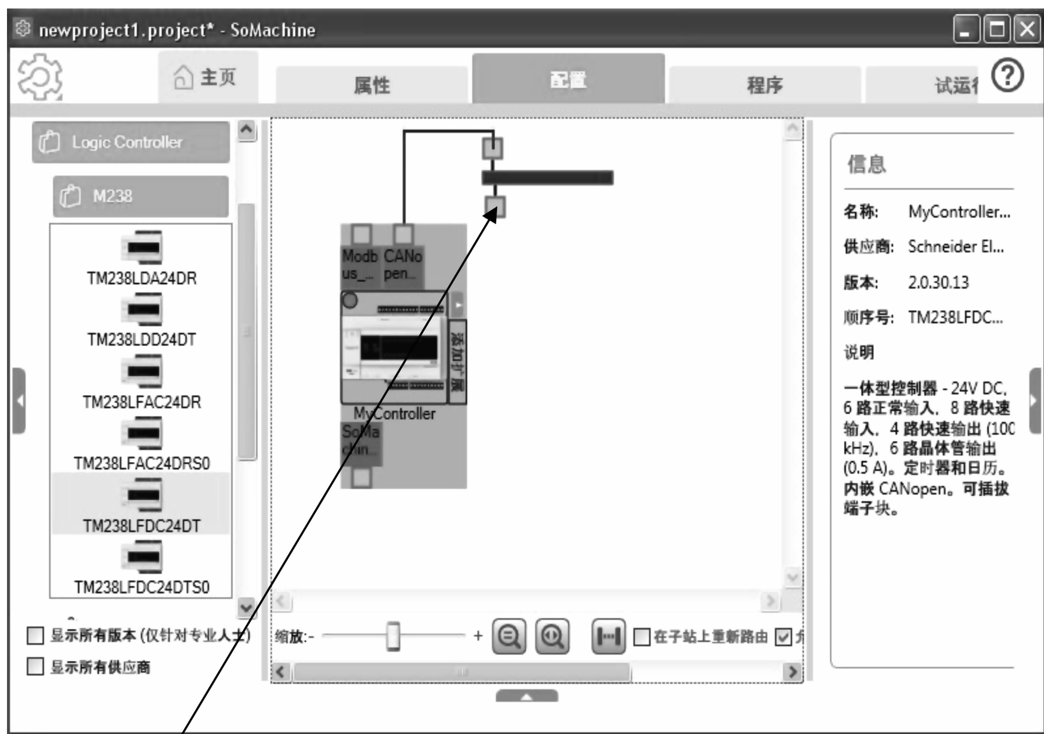


图 4-7　加入总线连接器

点击总线口，弹出添加设备界面如图 4-8 所示。

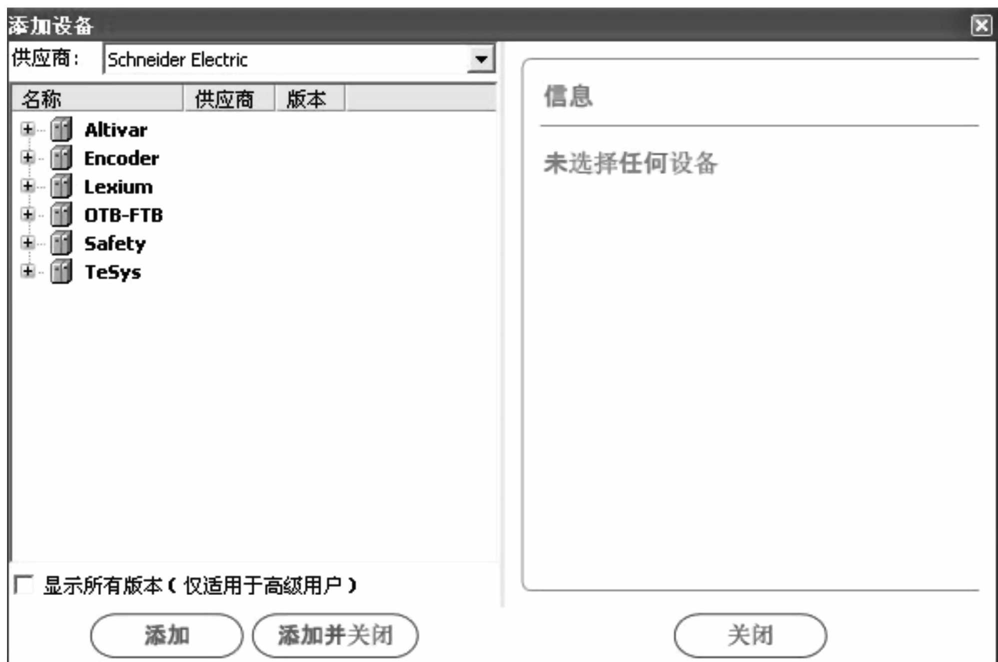


图 4-8　添加设备项列表

在这个设备项列表里，可以选择添加变频器（Altivar）、编码器（Encoder）、伺服驱动器（Lexium）、远程 I/O（OTB-FTB）、安全开关（Safety）、接触器（TeSys）等。如加入变频 ATV32 打开 Altivar，如图 4-9 所示选择 Altivar32。



图 4-9　添加变频设备

点击“添加”，则在 CANopen 总线口上加上了一台变频器，如图 4-10 所示。

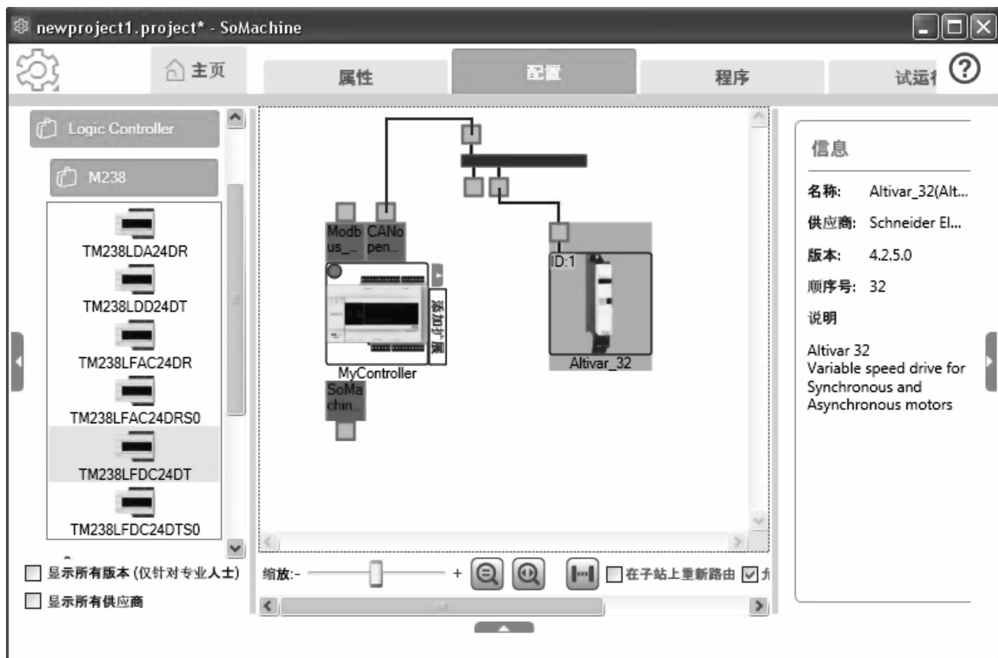


图 4-10　添加了一台变频器

在添加设备界面中，打开 Lexium，选择加入一台伺服 Lexium32A，如图 4-11 所示。



图 4-11　添加伺服 Lexium

点击“添加并关闭”，则又加入一台伺服 Lexium32A，如图 4-12 所示。

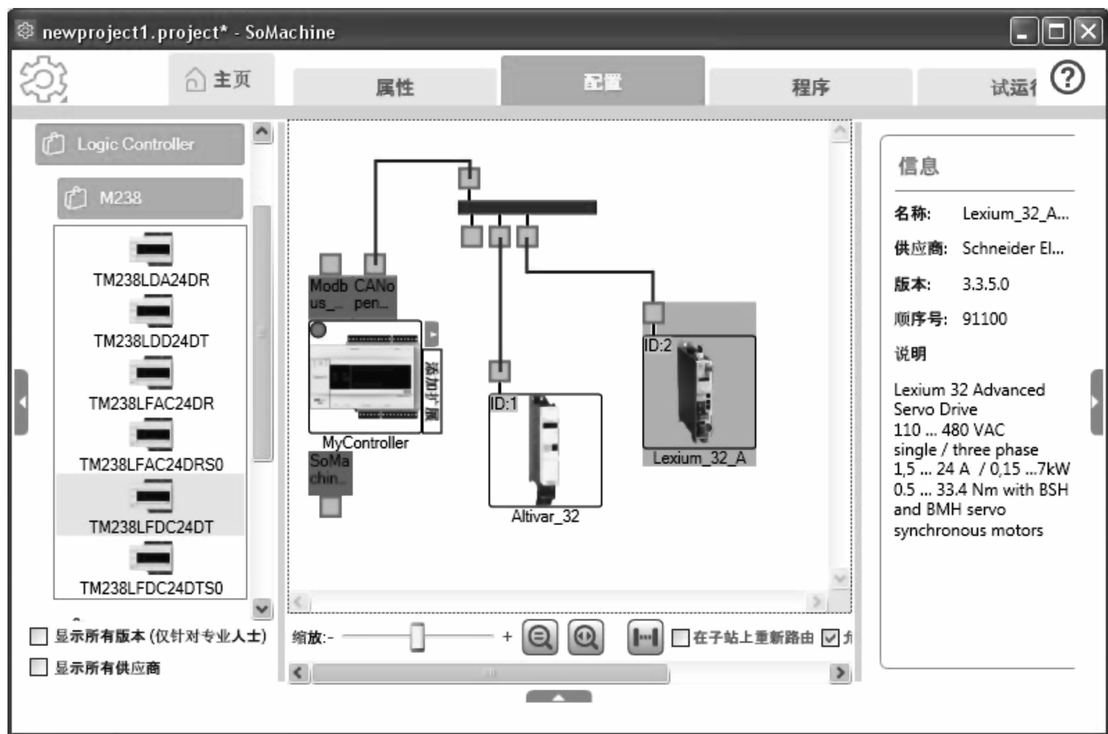


图 4-12　添加了一台伺服

同理，打开添加画面，还可以根据需要添加更多的控制设备和元件。当配置好控制系统的结构时，就可以进入程序，开始软件的设计。

4.3　以现有项目启动

有时新建一个项目，为了能够快速进入编程阶段，也可以借助已经搭好的控制结构的已有项目。如我们已经建立的项目 newproject1，它已经在 CANopen 总线上配置好一台变频器和一台伺服控制器，如图 4-13 所示。

我们把这个已有项目 newproject1 打开，然后以新的文件名 newproject2 存入设定的路径和文件夹，如图 4-14 所示，从而开始又一个新项目的设计。

这个新项目已经建立好了主体控制结构，当新加入项目信息后，就可进入配置，如图 4-15 所示。



图 4-13　以现有项目启动

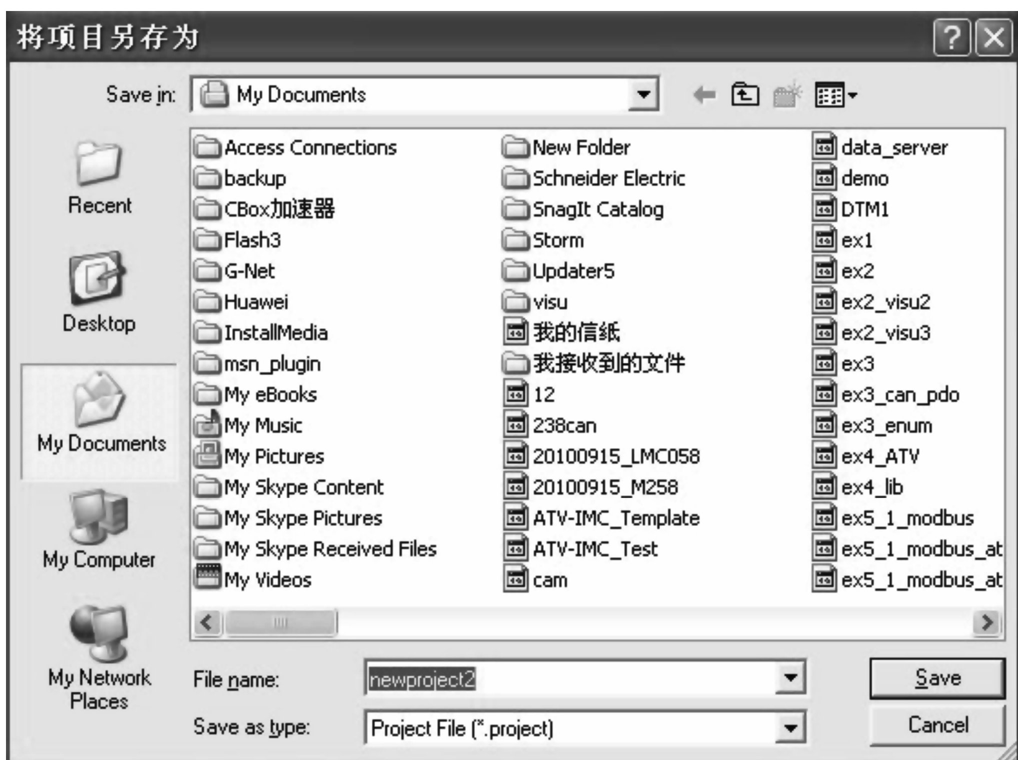


图 4-14　以新文件名存入



图 4-15　加入新的信息

在配置界面中，对已有的配置进行添加和修改，如图 4-16 所示。

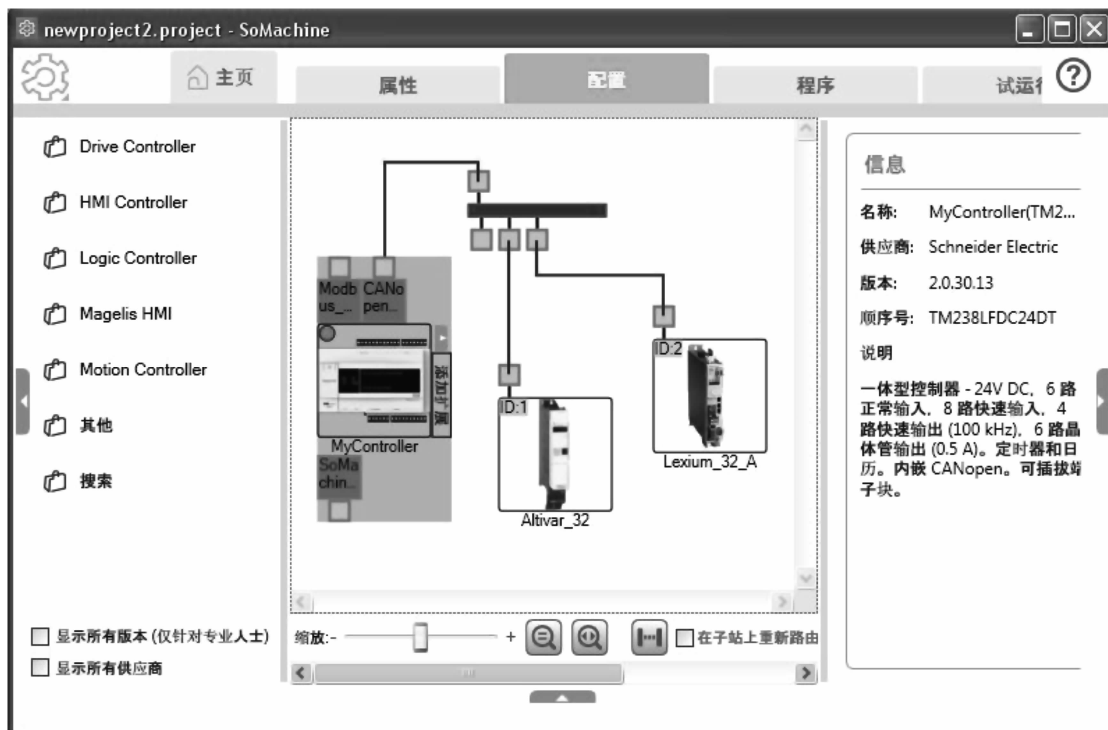


图 4-16　添加或修改已有的配置

当做好配置的添加和修改后，就可进入程序，进行应用程序的编制。在程序界面中，打开文件菜单，如图 4-17 所示可以对文件进行保存，也可以存入到制定路径下的文件夹内，还可以把项目涉及的文档压缩打包存入到指定路径下的文件夹。



图 4-17　文件处理

4.4　以模板项目启动

为了帮助搭建控制结构和编程应用，SoMachine 编程软件建立了 8 个不同结构配置，已成功使用并含有说明文件的项目供编程参考和快速启动。在主页面的创建新机器栏目下，点击使用 TVD 架构启动，如图 4-18 所示。

可以直接选择工作区列出的控制结构并打开，也可以用 TVD 架构查找器回答列出的问题并选择，然后打开建议的项目。例如打开 Optimized _ CANopen _ M238. project，并以新的项目名称 M238 _ optimized. project 存入指定路径下的文件夹内，如图 4-19 所示。

在属性页面中，修改项目信息或添加新信息后，打开配置如图 4-20 所示。

在这个配置中，已经做好了人机界面、变频、伺服和编码器的配置。打开程序，如图 4-21 所示，控制器应用程序已编好。



图 4-18　使用 TVD 架构启动



图 4-19　以模板启动一个新项目

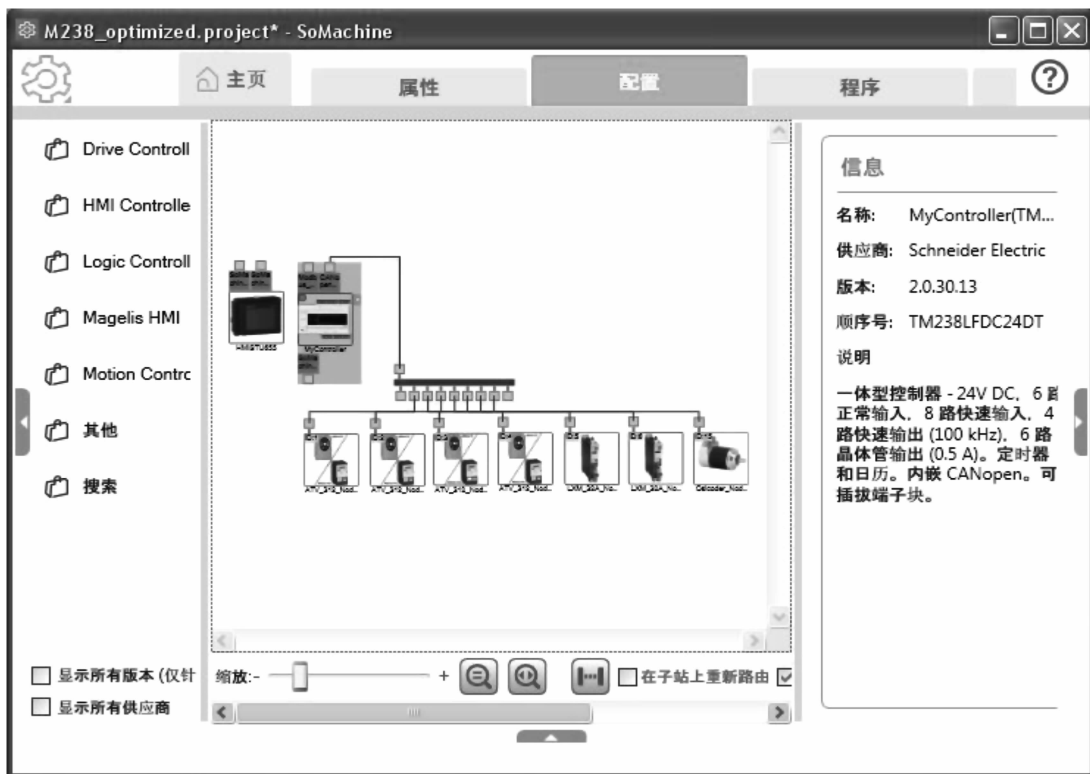


图 4-20　模板配置

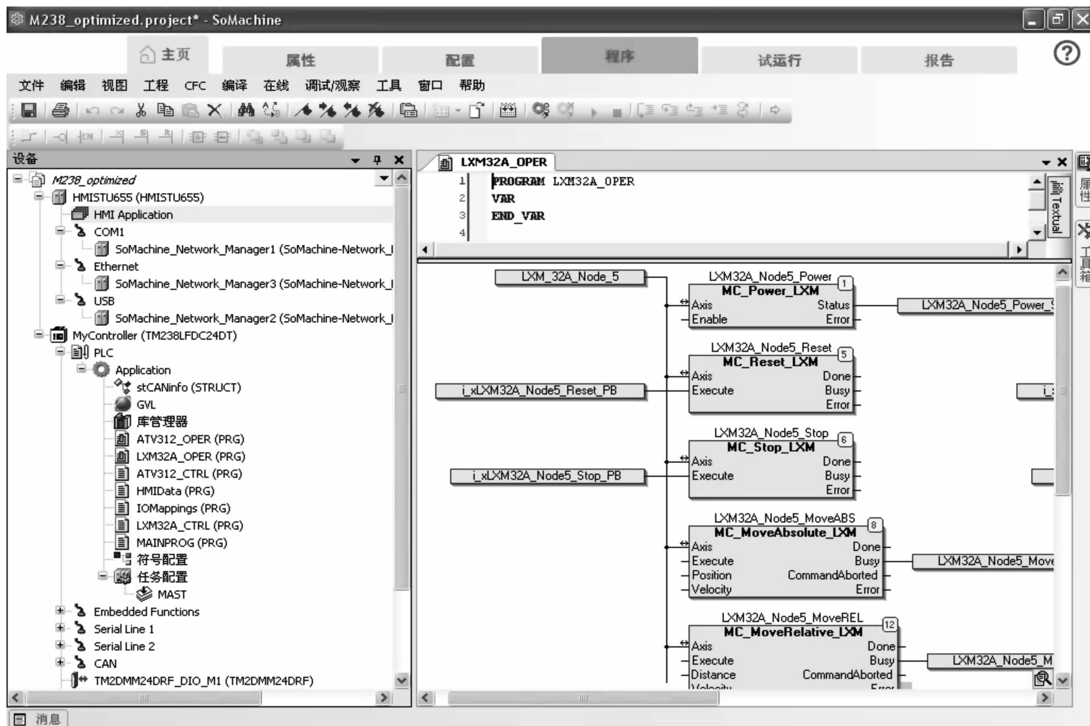


图 4-21　M238 逻辑控制器已编好的控制器程序

点击图 4-21 所示界面中的人机界面应用程序 HMI Application，得到界面如图 4-22 所示。

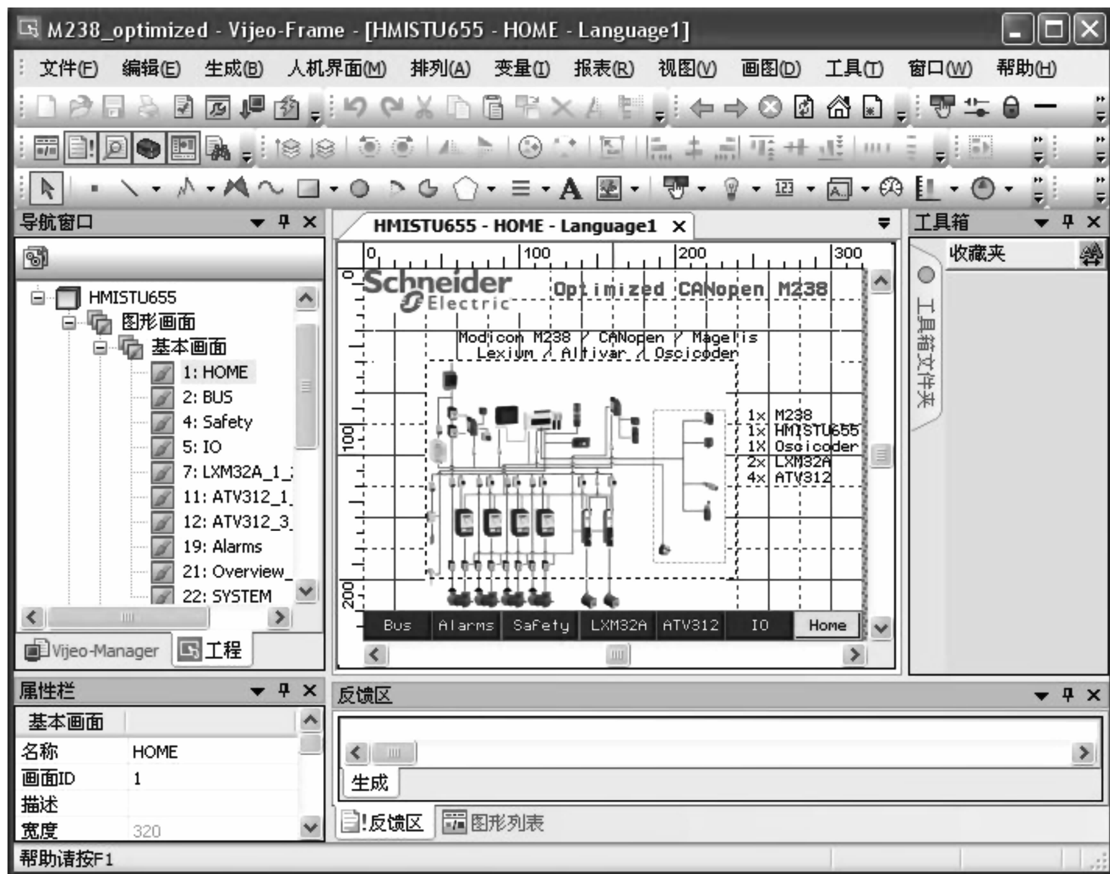


图 4-22 已编好的人机界面操作屏

此模板对配置的所有控制设备都编制了控制程序以及人机界面的操作和读写功能。所以，作为应用已经可以直接使用和操作。也可以在此基础上修改和增删。还可以对某些要用到的程序进行复制和粘贴。因此大大加快了开发程序的速度。在模板启动项里有不同的控制器和外围控制设备的模板供参考。

4.5 扩展模板的添加

在设计机器控制结构时，有的机器要求的控制点数较多，本体控制器上的 I/O 点数不够，所以就需要在本体上添加扩展模板。如对 M218 逻辑控制器，M238 逻辑控制器可以扩展 TM2 扩展模块系列扩展模板，I/O 点数可达到 248 点。对 M258 逻辑控制器，LMC058 运动控制器可以扩展 TM5 扩展模块，TM7 系列模板，I/O 点数可达到 2400 点。在添加扩展时，只要点击如图 4-23 所示的配置控制结构，本体控制器右边的添加扩展，就会弹出添加扩展模板列表，如图 4-24 所示。

选择要添加的模块，点击“添加”即可。

如选择数字输入扩展 TM5SDI2D，点击“添加”，如图 4-25 所示。

此时就可以看到，在配置结构中的扩展口加上了一个扩展板，如图 4-26 所示。

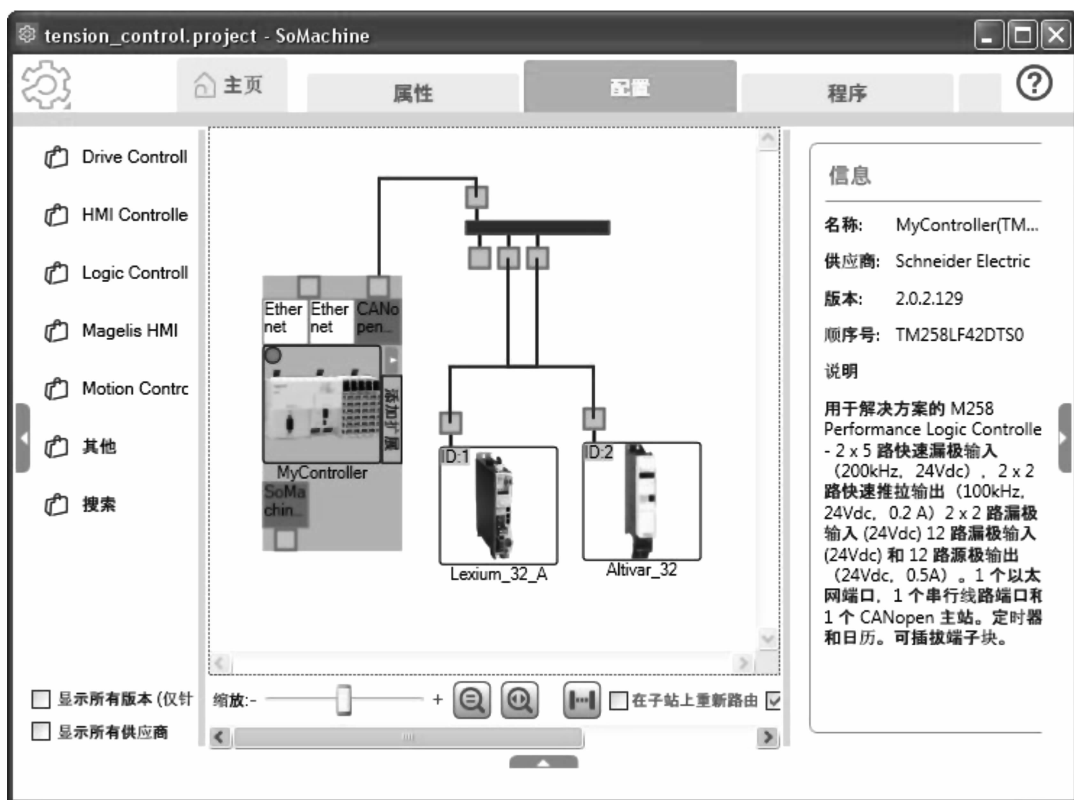


图 4-23　配置控制结构



图 4-24　添加扩展模板列表

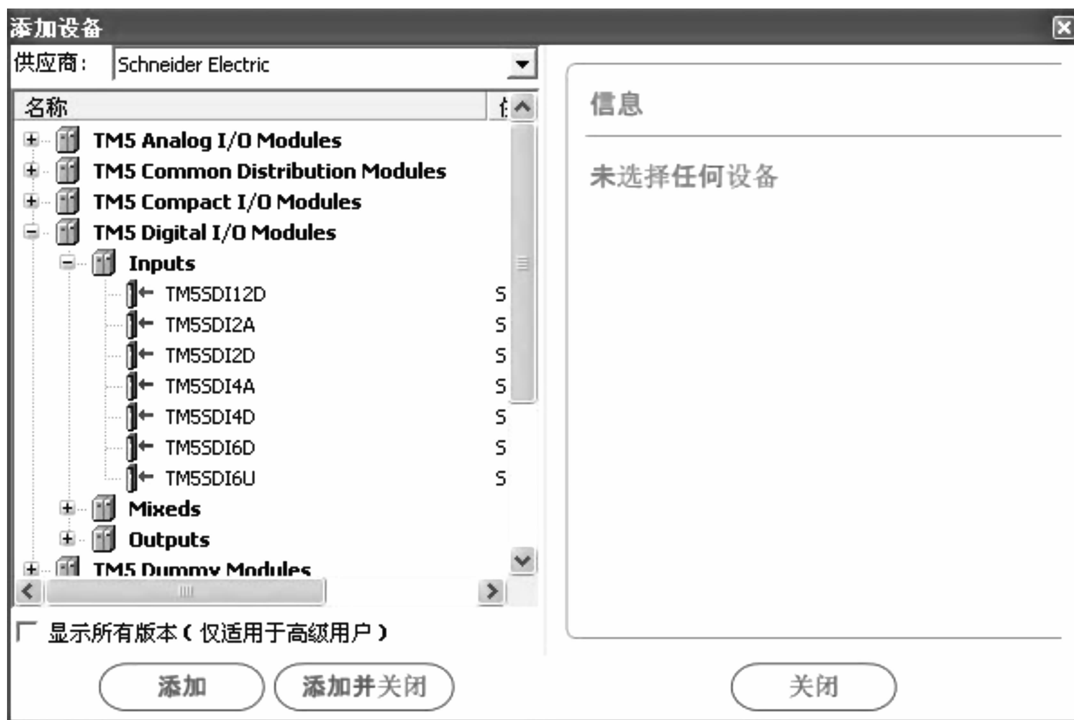


图 4-25　打开模板列表中的模块

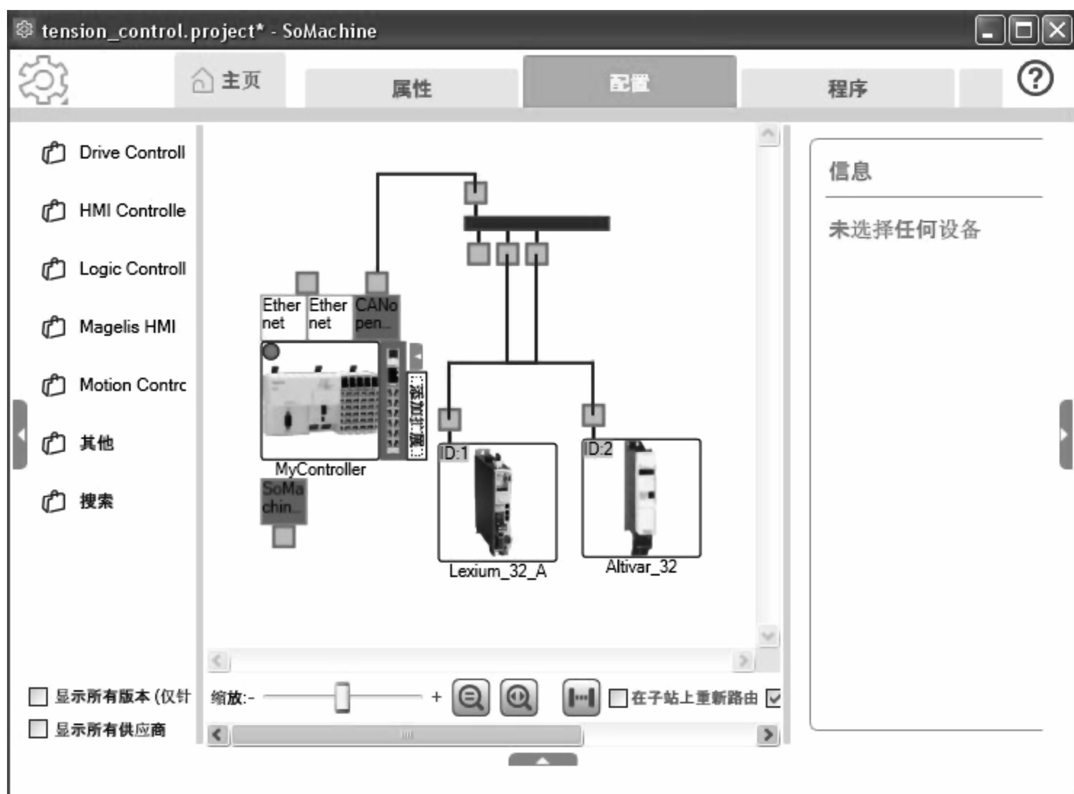


图 4-26　加入扩展模板

需要注意的是：添加过多的扩展模板会使本体电源的供电不足，造成编译错误，如图 4-27 所示，这样除了扩展应用模板外，当电源不足时，还要加入电源模板。



图 4-27　模板过多导致电源不足

电源模板的加入不能放在最后，它要求后面一定要有一块应用模板作为末端，如图 4-28、图 4-29 所示。



图 4-28　电源模板放在末端引起错误

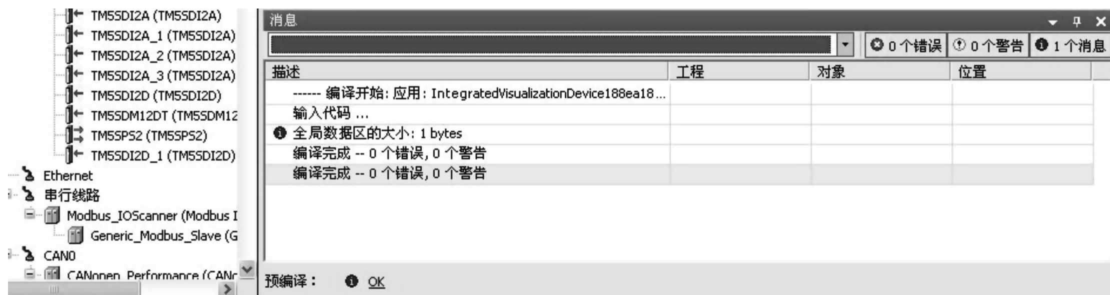


图 4-29　正确放置电源模板

第 5 章　编辑控制器程序

5.1　概览

PLC 的编程语言在过去十几年的发展中，厂商各自开发基于梯形图编程、C 语言、BASIC 语言以及其他各式各样的独特的开发语言。由于这些语言的不同，导致使用者不得不花费大量时间来学习各种编程语言。正是这种原因，国际电工委员会（International Electrotechnical Commission，IEC）早在 1990 年就提出了 PLC 编程开发标准 IEC 61131。而在 1993 年提出的 IEC 61131-3（Part 3）就规定了 PLC 的编程语言标准。

SoMachine 编程软件采用了符合这个标准的 6 种编程语言。它们是顺序功能图（Sequential Function Chart，SFC）语言和 5 种可共用的编程语言：

1）梯形图（Ladder Diagram，LD）；

2）功能块图（Function Block Diagram，FBD）；

3）连续功能图（Continous Function Chart，CFC）；

4）结构文本（Structured Text，ST）；

5）指令表（Instruction List，IL）。

而 SFC 主要用于偏重顺序过程的控制，如一台往复周期动作的机器。这个特性同样也适合于控制不同工艺状态下动作的机器。即监控机器的各种工艺状态，如起动机器，停止机器，机器的报警状态等。

当我们做完项目的硬件配置后，就可以进入编程阶段了。点击程序，进入到编程阶段。图 5-1 所示为编程画面。

在这个程序界面的设备栏目下，可以和所配置的硬件平台对应上。例如 M258 逻辑控制器，你可以看到程序的编程环境 Application，也可以看到本体的硬件资源，以及以太网通信口和串行通信口，CANopen 通信口。

图 5-2 所示的设备窗口由 4 个主要部分组成。双击这些部分，就可以打开它的树状结构进行设备的组态或软件的编程。这 4 个部分是：

1）控制器：PLC 的类型和组态。

2）应用程序：PLC 程序的所有组成。

3）内置功能：如建立在 M238 逻辑控制器的功能。

□I/O（fast & normal I/O）I/O 点（高速和普通点）；

□HSC high speed counter（高速计数器）；

□PTO（脉冲串）和 PWM（Pulse Train Output，Pulse Width Modulation）（脉宽调制）。

4）通信：与外部设备的通信口。

□Advantys；

□CAN devices；

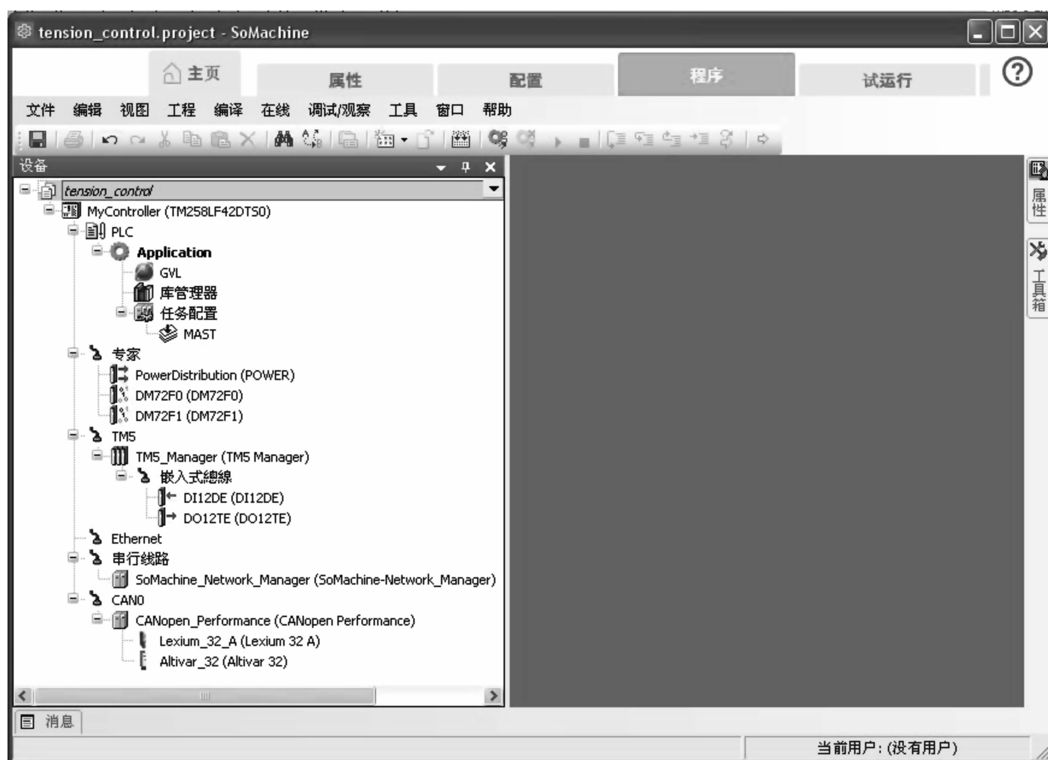


图 5-1　编程画面

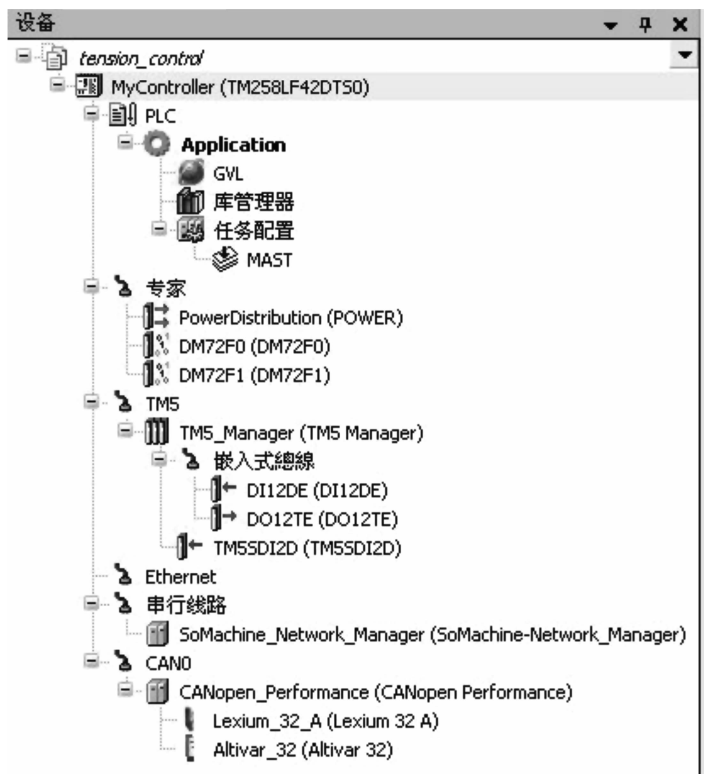


图 5-2　软件编程和硬件组态

□Servo & Stepper drives；

□etc.　...

双击图 5-2 所示的 MyController 就可以建立计算机和 M258 逻辑控制器的通信。

首先添加网关，然后扫描网络，如图 5-3 所示。



图 5-3　添加网关

在网络中找到控制器，如图 5-4 所示，点击设置活动路径。



图 5-4　找到控制器

然后按照弹出的说明框，按“ALT + F”激活通信路径，如图 5-5 所示。

然后点击连接进入控制器，如图 5-6 所示。

如图 5-4 所示，控制器的其他栏目还有：

1）应用：显示 PLC 应用列表。

2）PLC 设置：运行 I/O 的应用名，当在停止模式时的输出状态。

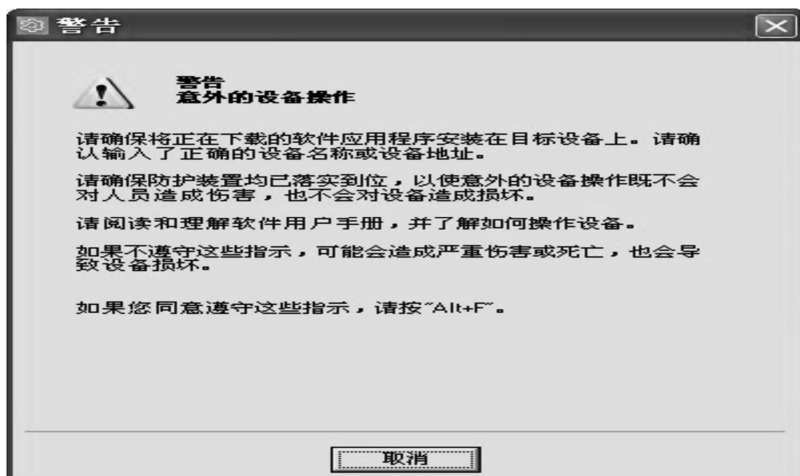


图 5-5　激活路径



图 5-6　进入控制器

3）服务：设置 PLC 时钟，显示固件版本，引导版本。

4）状态：PLC 运行状态和扩展总线状态。

5）信息：连接的 PLC 控制器信息。

程序部分包含下列内容：

□Application

－所有 PLC 应用程序。

□GVL（Global Variable List）全局变量列表

－全局变量解释。在本项目中的所有程序均可见；

－每个项目最多有 3 个全局变量列表。

□库管理器

－在本项目中添加库或建立用户自己的应用程序库；

－程序库提供下列在运行中执行的项目。

自动控制设备的操作系统：

☐功能和功能块；

☐定义的数据类型；

☐全局变量；

☐可视界面；

☐程序（PRG）和功能（FUN）。

－POU（Program Organizational Unit）是程序的组织单元，用来建立运行程序。

功能是在当前项目中建立的附加功能。可以在任何 6 种编程语言中建立。

☐任务配置

－在控制项目中执行程序。

MAST-主项目，启动一个新项目时系统自动建立。

将鼠标移到 Application，点鼠标右键，然后选择添加对象，如图 5-7 所示。

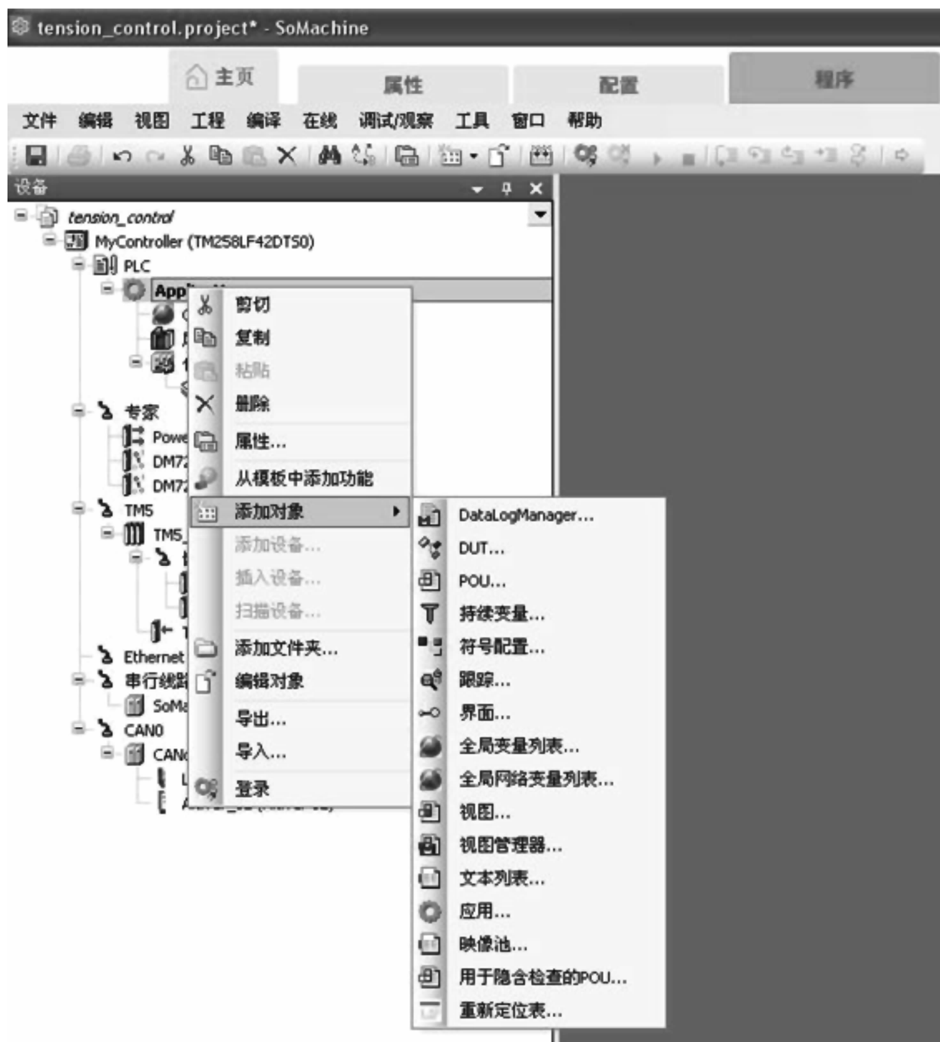


图 5-7　添加对象

在添加对象的列表中，显示出了可添加的很多功能，根据需要可选择添加。目前，我们先选择添加对象中的 POU（程序组织单元）（Program Organizational Unit）。这时弹出画面如图 5-8 所示，创建一个新的 POU。



图 5-8　创建新的 POU

在生成 POU 时，可以有 3 种选择。

1）程序：它由一系列指令、功能块和功能组成，可以直接执行。建立后形成后缀（PRG）文件。

2）功能块：由一系列输入指令和输出指令封装的有设定功能的模块，不能单独执行。只能在程序中调用。建立后形成后缀（FB）。

3）函数：由多个输入形成一个反馈输出的功能模块。不能单独执行，只能在程序中调用。建立后形成后缀（FUN）。

通过点击实现语言的下拉菜单，可以为 POU 的编程选择 6 种编程语言，如图 5-9 所示。

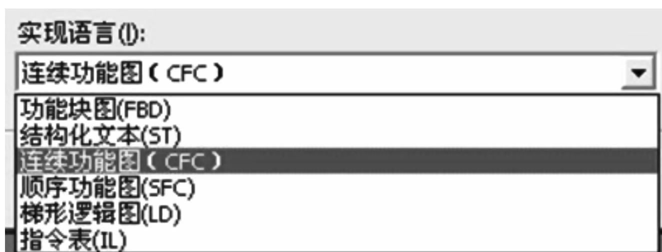


图 5-9　选用编程语言

选择好编程语言，就可以进入编程操作。

5.2　POU 程序创建

当我们完成控制器硬件的组态，就可以开始设计程序的编制。控制器运行的程序是由组织单元来管理的。因此，我们要建立组织单元 POU，如图 5-10 所示。打开添加对象，添加 POU，然后选择连续功能图（CFC）。点击打开，则在设备栏目下的 Application 下建立了 POU（PRG）。并且右边开放出了编程区域。

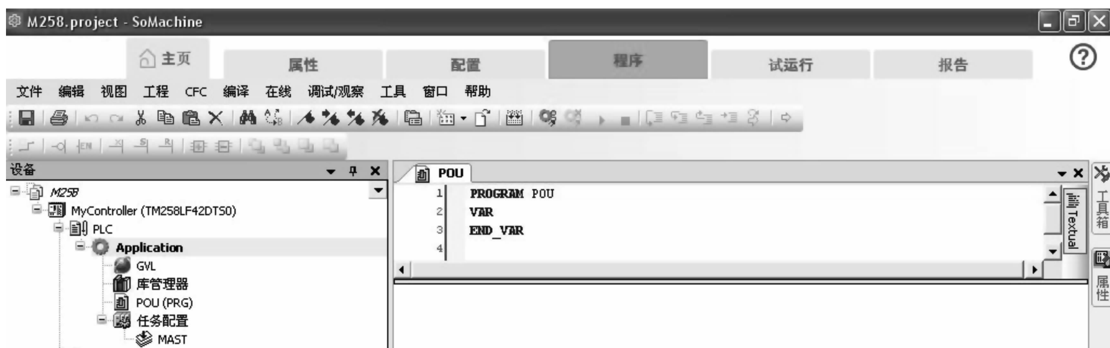


图 5-10　建立 POU 编程区（CFC）

在编程区，我们借助工具箱来编程。首先，打开工具箱，选择一个运算块，如图 5-11 所示。



图 5-11　打开工具箱

然后把功能块拖放在编程区，如图 5-12 所示。

点击图 5-12 中功能块上的问号，会出现一个白框。

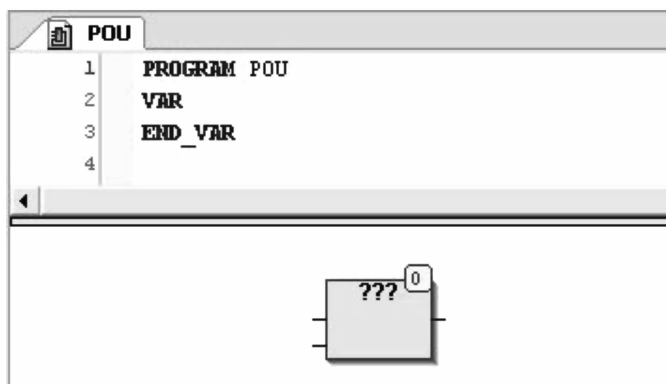


图 5-12　拖到编程区域的功能块

点击图 5-13 功能块上的白框，弹出输入助手，如图 5-14 所示。输入助手下有各种已经做好的标准功能块、模块调用、关键字和转换操作符。选择所需要的，按确认键。

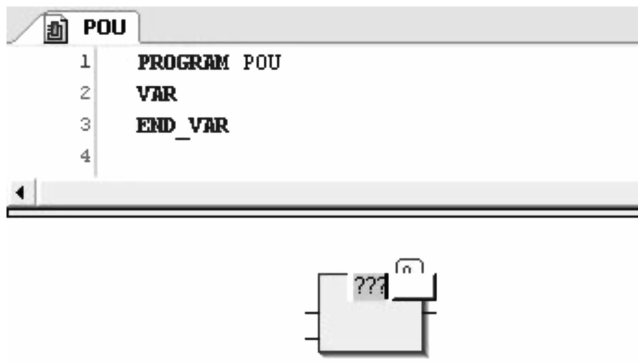


图 5-13　功能块辅助输入图标



图 5-14　输入助手

例如，选择相加“ADD”，然后确定，如图 5-15 所示。



图 5-15　输入助手显示的关键字

在编程区域加上了一个相加的功能块，如图 5-16 所示。

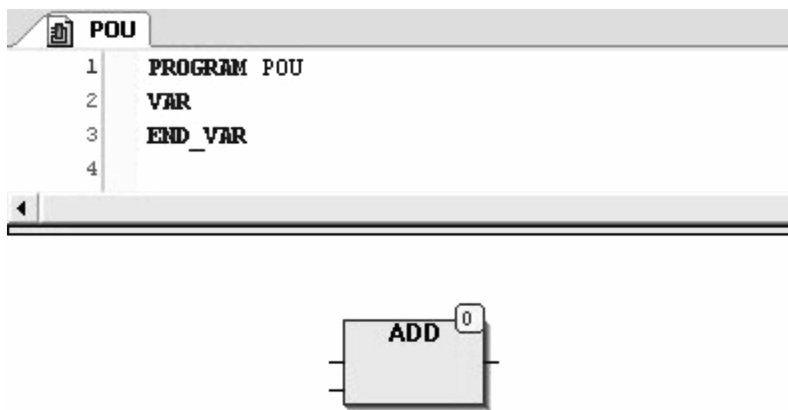


图 5-16　相加功能块

在功能块上写一个输入变量并定义数据类型，如图 5-17、图 5-18 所示。

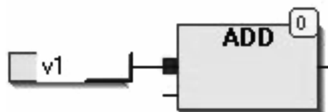


图 5-17　定义输入变量



图 5-18　定义变量数据类型

选择变量的数据类型为 INT，按确定。则在变量定义栏目出现定义好的变量列表，如图 5-19 所示。

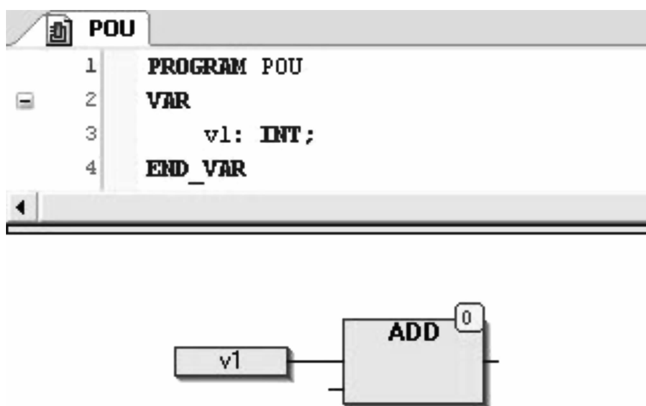


图 5-19　定义好的变量

同样方式，我们加入另两个变量 v2，s1，如图 5-20 所示。

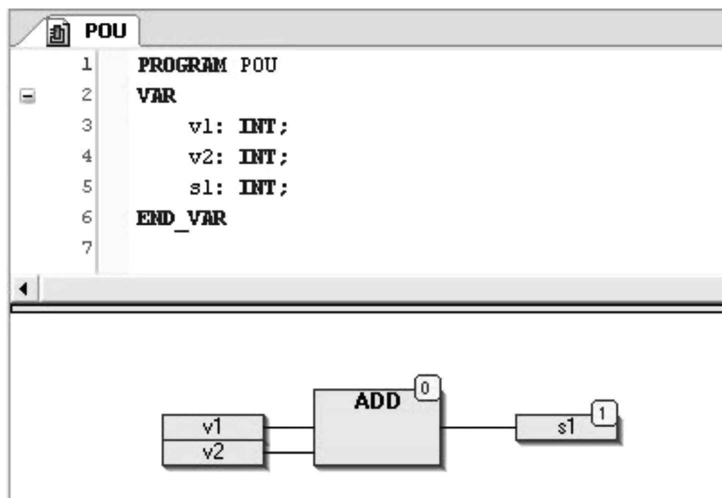


图 5-20　CFC 语言编制的一个 POU

这样，我们就采用 CFC 语言编了一个 v1 + v2 = s1 的程序。要使这个程序执行，我们还需要把程序添加到任务配置的 MAST 如图 5-21 所示。



图 5-21　任务配置的 MAST

点击 MAST，图右边打开 MAST 画面，点击添加 POU，弹出输入助手，如图 5-22 所示。



图 5-22　添加 POU 输入助手

选择建立的 POU，按确定，如图 5-23 所示。

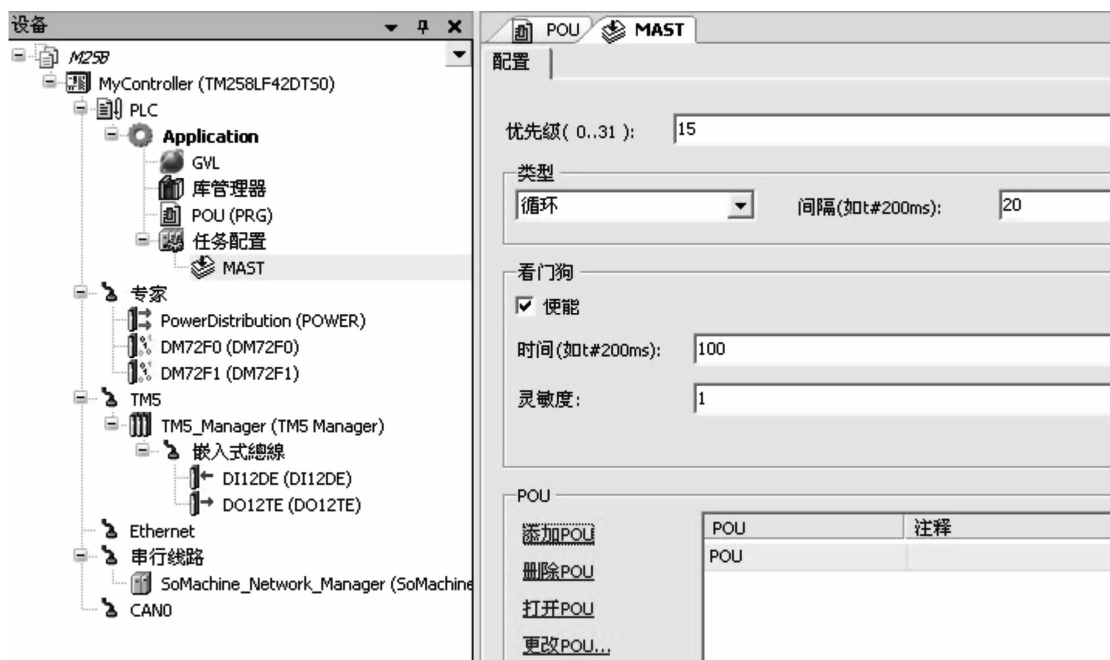


图 5-23　添加 POU 完毕

添加完毕后，就可以编译程序。按“F11”，或点击图标，如图 5-24 所示。



图 5-24　程序的编译

程序编译状况，可以在消息栏目中显示出来，如图 5-25 所示。

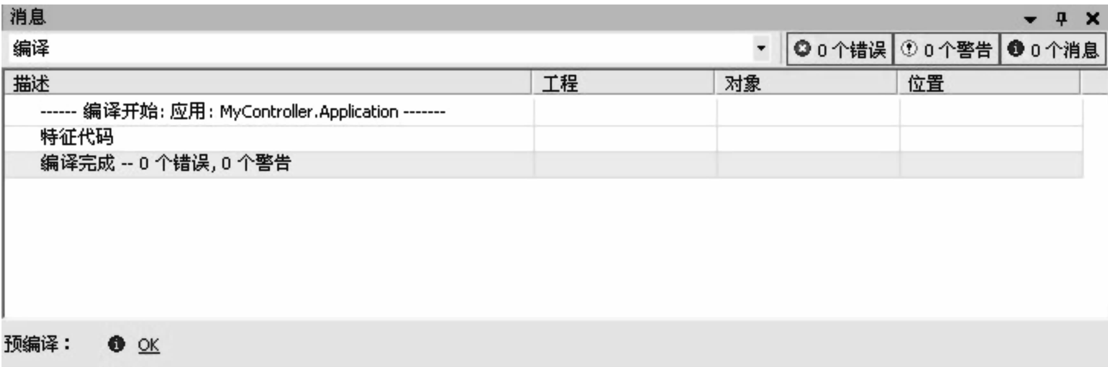


图 5-25　编译信息栏目

5.3　任务配置

编辑好的程序，需要纳入执行的列表，才能发挥功能。如果不在执行列表，就意味着不运行该程序。在控制器中这个执行列表功能就是任务配置，如图 5-26 所示。

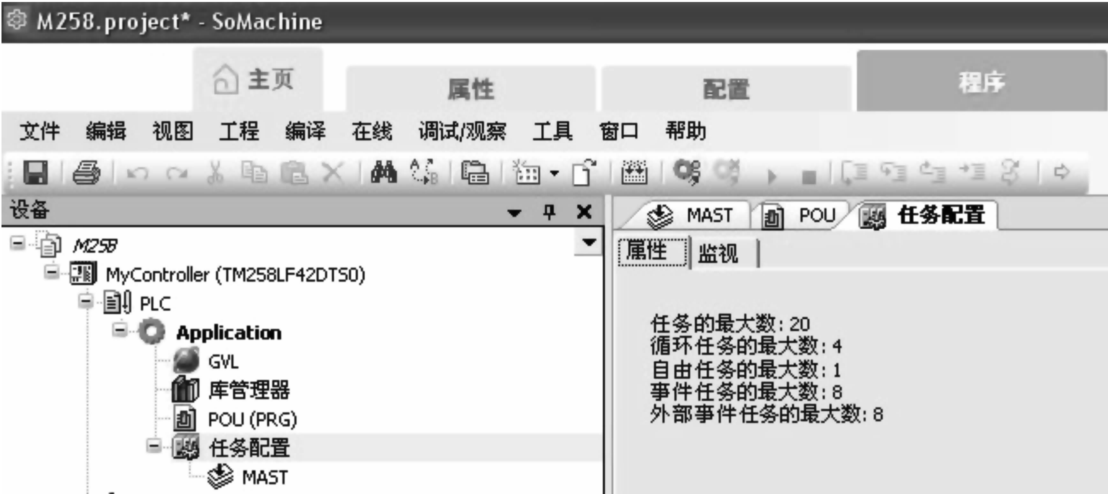


图 5-26　任务配置

双击任务配置，会显示出属性和监视。在属性栏目，列出了任务的 4 种属性和可执行的数目。在监视栏目，你可以观察到任务的运行周期和次数，以便改进程序和评估控制器执行性能能否满足工艺要求。在任务栏目下，自动生成一个 MAST，用来定义任务的属性，如图 5-27 所示。

双击 MAST 打开配置，如图 5-27 所示。最先看到的是执行的优先级。级别最高为‘0’。最低为‘31’。在类型选项中，有 4 种类型可选。

1）自由运行：如图 5-27 所示，程序的运行从头到尾执行，由于每个周期的运行时间不一样，所以每次对程序的执行周期也是不一致的。

2）循环：程序从头至尾循环运行，但刷新程序的周期是一致的，并且需要定义。一旦程序的运行时间超过刷新周期，则报警。如图 5-28 所示，程序循环时间设定为 20ms。

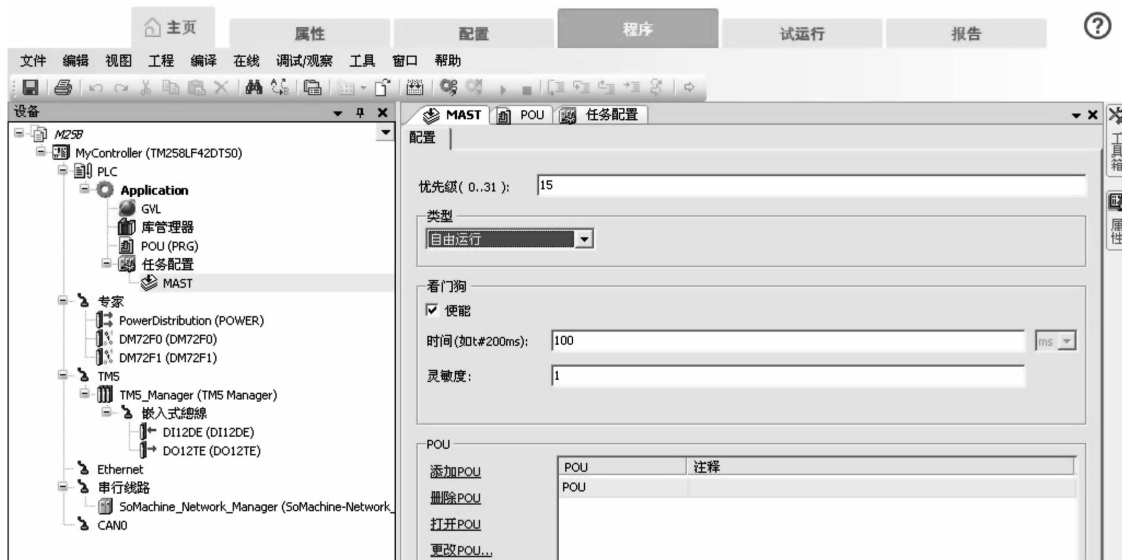


图 5-27　任务属性的定义

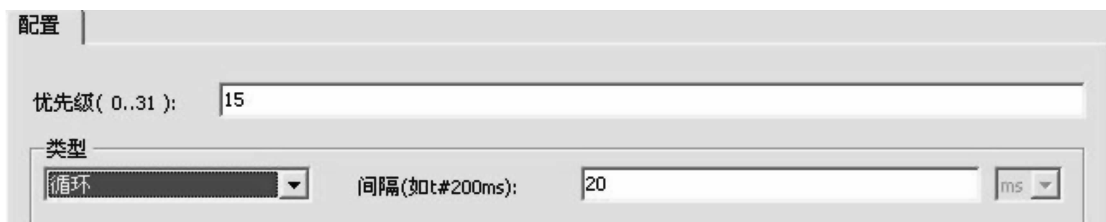


图 5-28　程序循环运行

3）事件：在程序中定义一个变量，当程序运行时，把这个变量的状态变化作为一个触发条件而引起的另一个程序的执行，就是一个事件。引起运行的程序就是事件程序。这个变量一般是程序内部定义的变量。如图 5-29 所示，定义了程序中的一个布尔量 s2，当 s2 为真，启动这个事件程序。

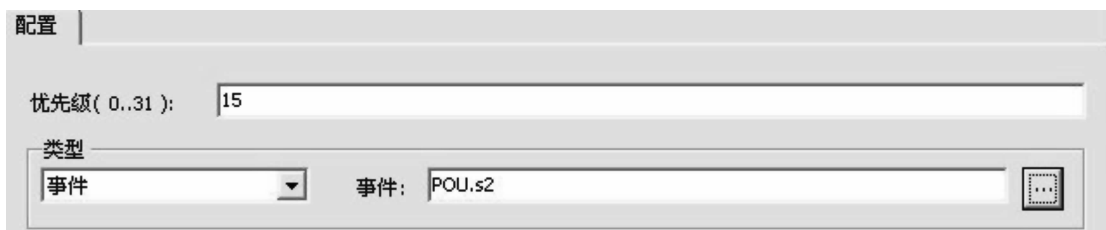


图 5-29　事件程序的定义

4）外部的：与事件一样，这个程序也定义为事件程序。但不一样的是：触发执行的事件来自外部，例如控制器的外部输入点；外部计数脉冲的输入到达或超过定义的阈值等。如图 5-30 所示，外部事件触发为输入点 I0。

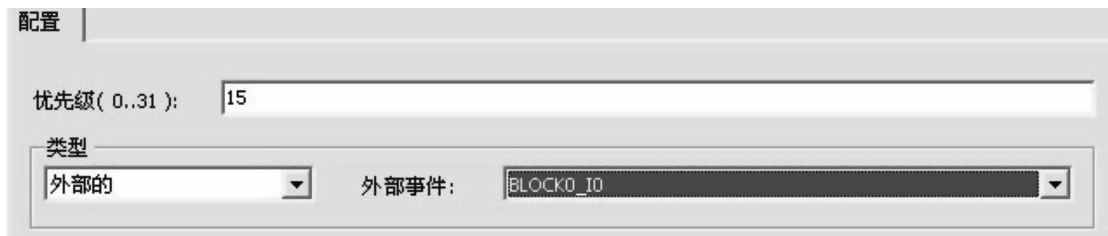


图 5-30　外部事件程序的定义

5.4　PLC 的程序仿真

对于一个已经建立好的 POU 程序，并且把它添加到了任务配置中，我们就可以进行程序的离线仿真。例如，仿真图 5-20 的程序，打开在线菜单，选择仿真后，下部信息栏会出现仿真提示，如图 5-31 所示。

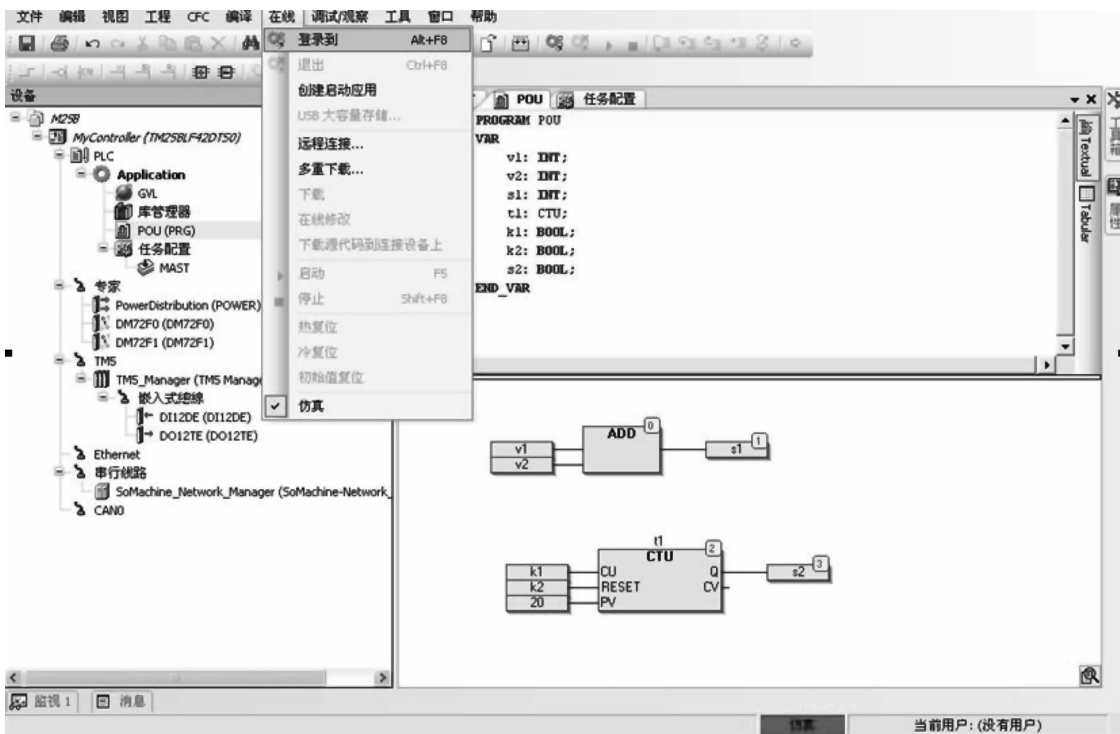


图 5-31　程序仿真模式

然后，再点击在线-登录到，则出现如图 5-32 所示画面。按“是”进入下载程序。

编译成功后，进一步确认下载，如图 5-33 所示，按“是”进入虚拟控制器，进行仿真。

点击图 5-34 所示在线菜单的启动或按“F5”，运行仿真。这时，打开 POU 程序，给变量 v1，v2 赋值，相应的就会看到变量 s1 的运算结果，如图 5-35 所示。



图 5-32　仿真程序下载

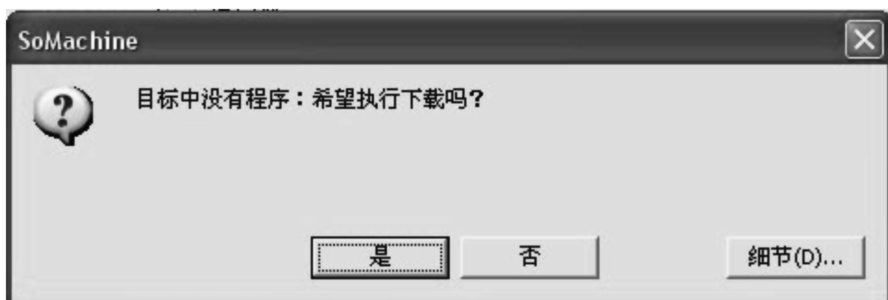


图 5-33　再次确认下载

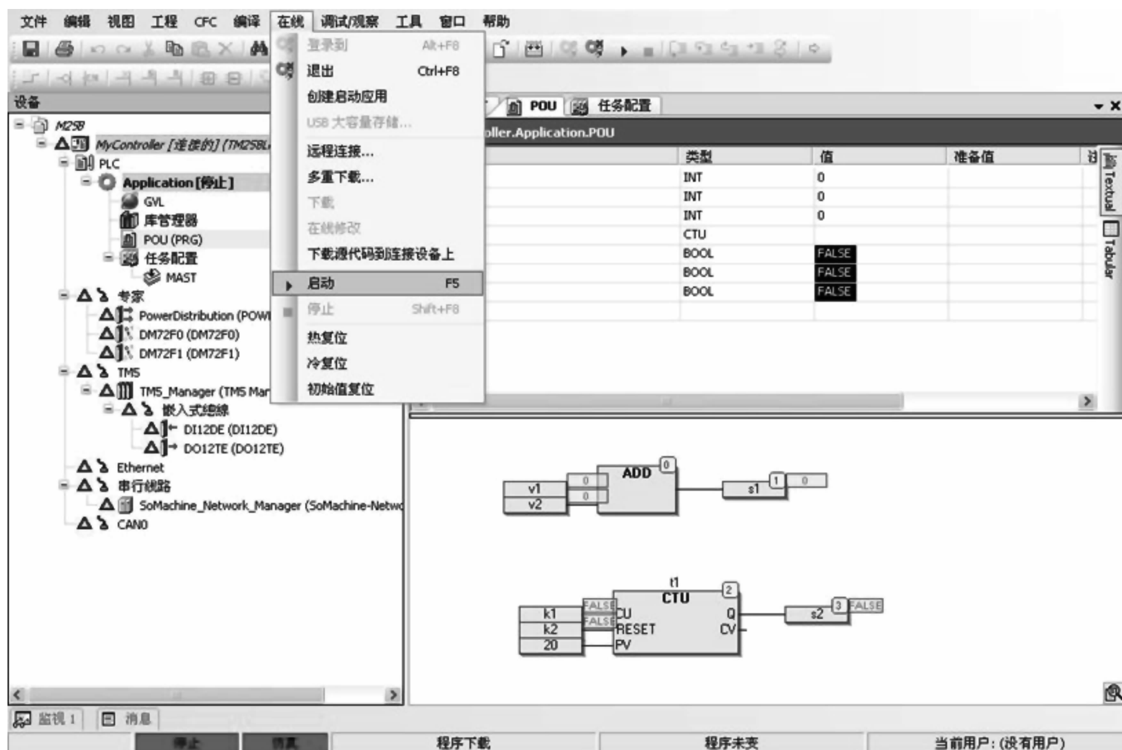


图 5-34　进入仿真状态

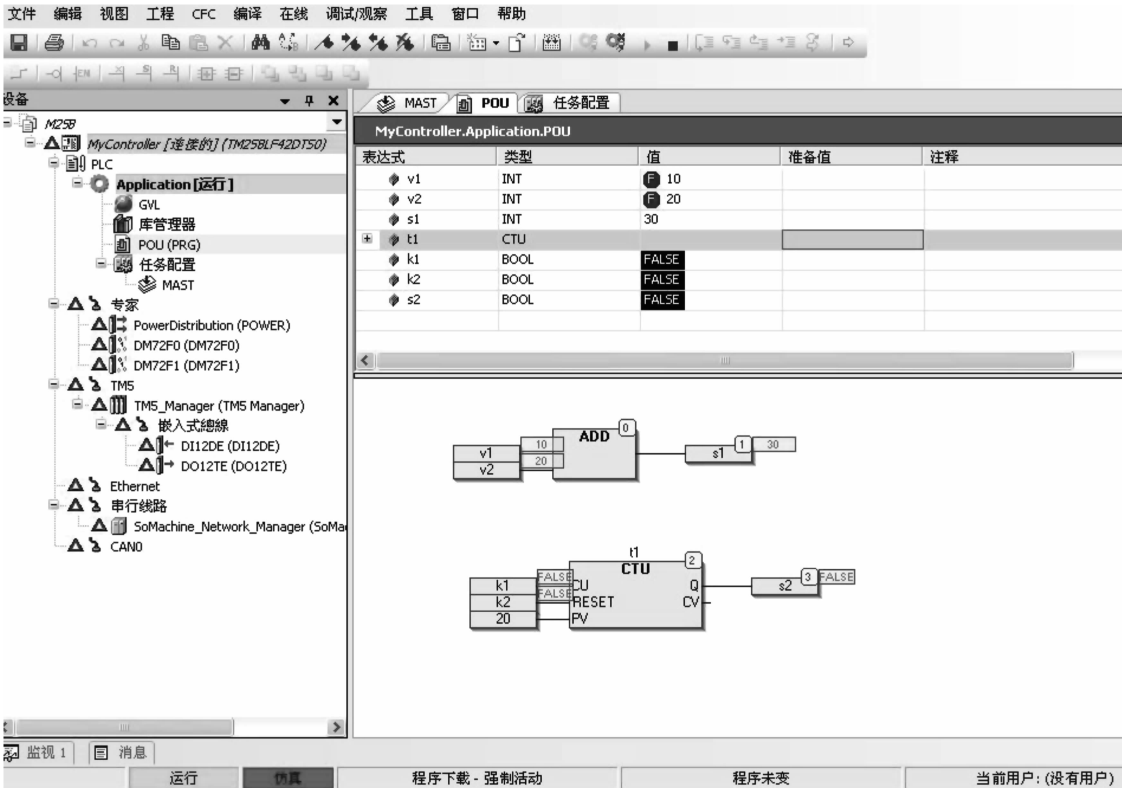


图 5-35　运行仿真

一些有用的快捷键：

- 登录“Alt + F8”/退出“Ctrl + F8”；
- 启动“F5”；
- 停止“Shift + F8”；
- 断点“F9”；
- 写数值“Ctrl + F7”；
- 强制值“F7”；
- 编译“F11”。

5.5　CoDeSys 编程语言

SoMachine 的软件编程是基于 CoDeSys（Controlled Development System）编程软件的，符合 IEC61131-3 标准，共有 6 种编程方式。这 6 种编程方式极大地方便了功能的实现。使设计人员对机器控制的设计和编程已经不局限在一种编程格式，也不拘泥于只对逻辑状态进行编程。他们根据工艺要求而采用顺序流程图（SFC）方式规划结构，采用结构文本（ST）

方式进行复杂工艺运算和调节计算，采用梯形图（LD）方式处理各种逻辑和工艺过程逻辑计算，采用功能图（FBD）方式进行同一功能的反复使用和对通信功能的搭建。因此，在建立一个 POU 时，我们可以选择实现程序的语言。

5.5.1 POU ST 结构文本编程方式

这种方式类似于高级语言，可以方便地编写复杂运算程序以及各种跳转和循环。在建立 POU 时，选择实现语言为结构化文本（ST），如图 5-36 所示，建立的 POU 为 POU _ ST。

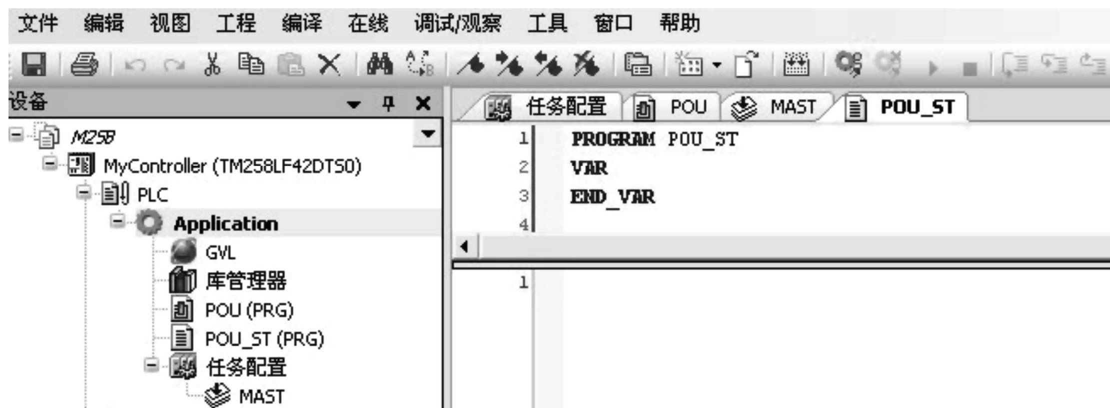


图 5-36　建立 POU _ ST

在程序变量定义区，我们定义输入点 I0，I1 和输出点 Q1 为布尔量。在编程区，我们写出输出变量 Q1 和输入变量 I0，I1 的逻辑关系，如图 5-37 所示。

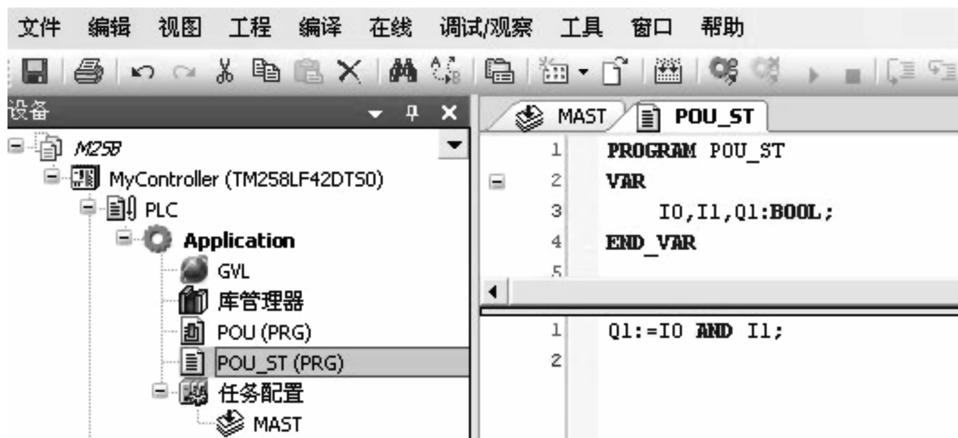


图 5-37　结构化文本编程

在采用文本编程时，我们也可以用快捷键“F2”，或在编程行点击鼠标右键来调出输入助手，如图 5-38、图 5-39 所示。

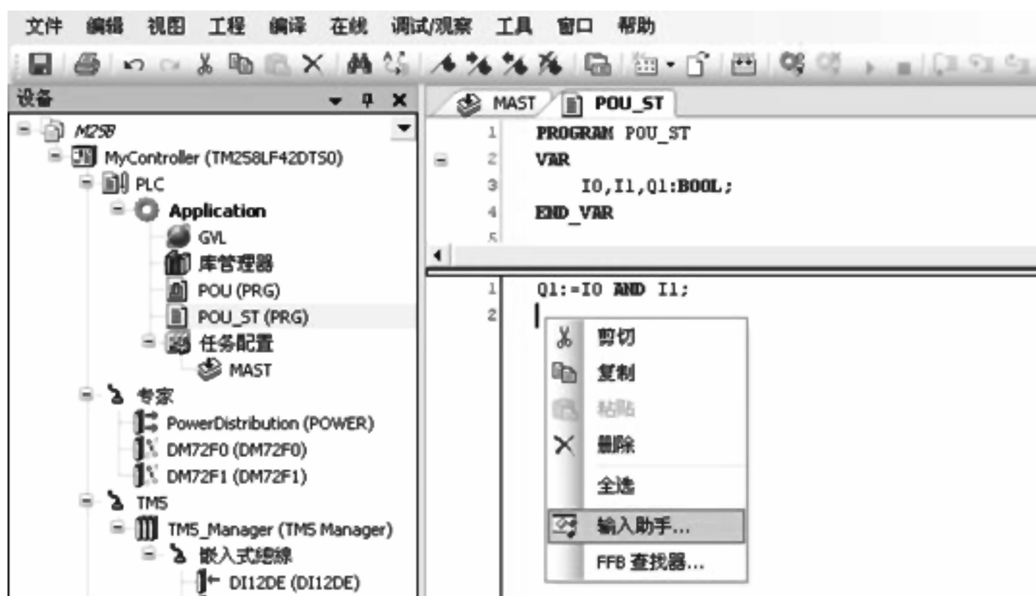


图 5-38　点击鼠标右键，选择输入助手

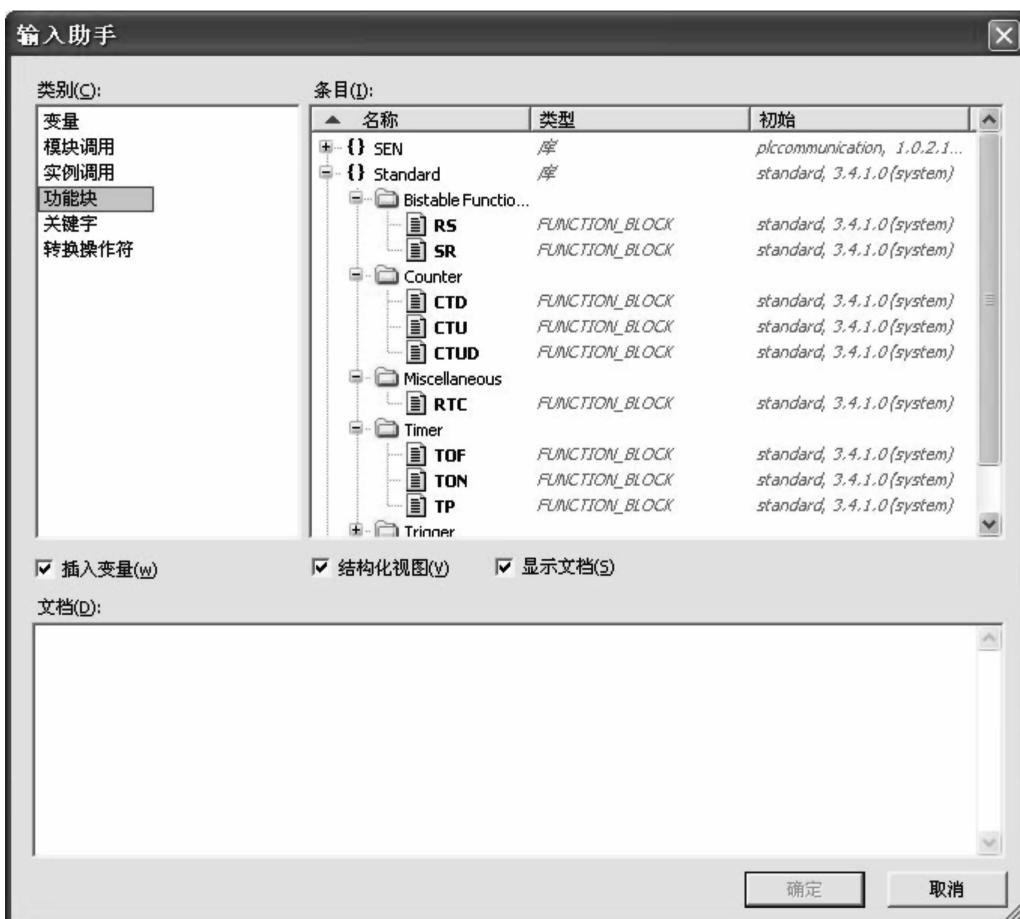


图 5-39　打开输入助手，选择所需功能块或变量

通过输入助手，我们可以调用已经做好的功能块、变量、命令关键字和操作转换符。

例 1：编辑一个延时输出功能。要求延时 6s，输出打开。

首先，按“F2”打开输入助手，点击功能块，选择“TON”，如图 5-40 所示。



图 5-40　打开输入助手，选择功能块

弹出功能块自动声明，需要对调用的功能块命名，如图 5-41 所示。



图 5-41　命名确认功能块

命名这个功能块为 t1，按“确定”，如图 5-42 所示。



图 5-42 命名功能块名称

程序行出现调用的功能块 t1，如图 5-43 所示。

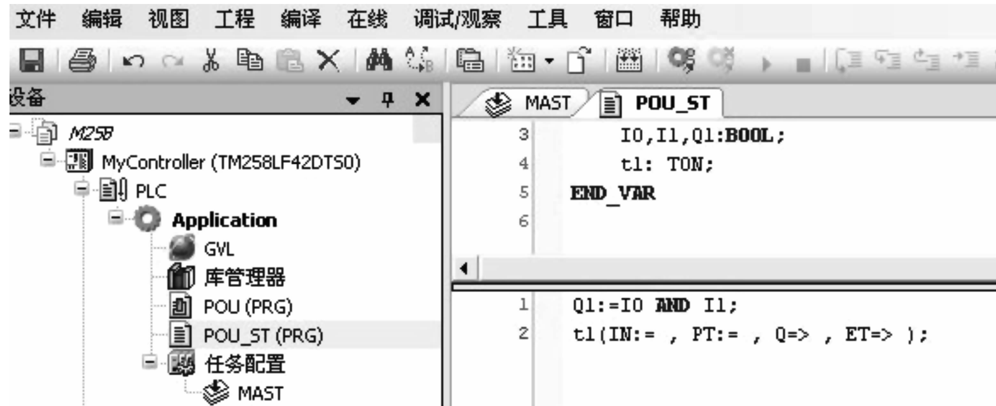


图 5-43 在程序行写入 t1 功能块

在程序中，可以给功能块的输入和输出定义变量，也可以通过仿真来观察程序执行情况。我们把编辑的程序 POU _ ST 加入到任务配置下的 MAST，并且定义 POU _ ST 的执行类型为循环，循环时间为 20ms，如图 5-44 所示。

设定 PT 为 6s，执行仿真，使 IN 为“真”。则延时开始，如图 5-45 所示。

延时 6s，输出 Q 为“真”，如图 5-46 所示。

例 2：条件语句的使用

格式 1：IF 表达式 THEN 语句块

END _ IF 结束 IF 语句

```
(* if ...then *)
IF V1 AND V2 THEN Q1:=TRUE;

END_IF
```

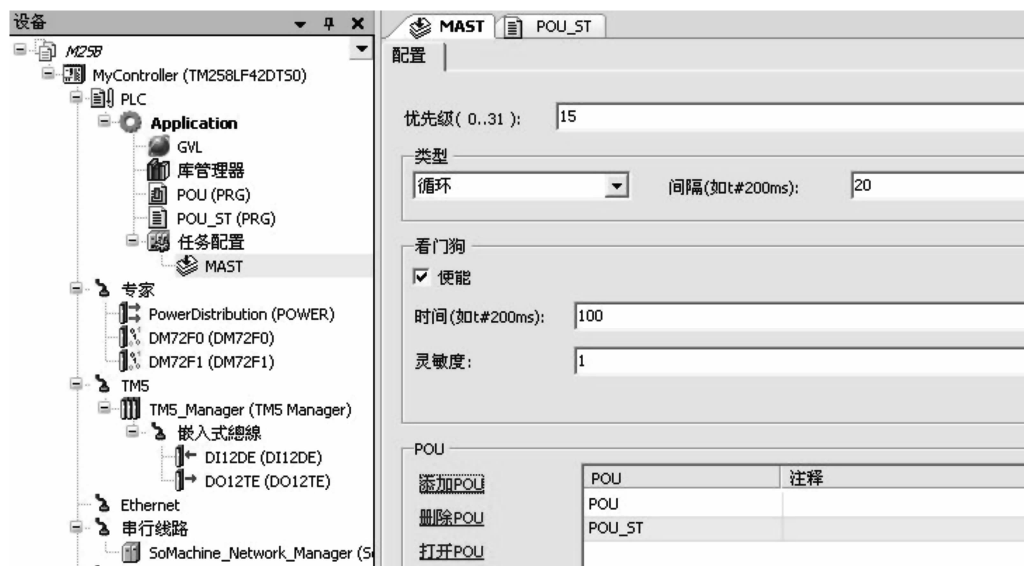


图 5-44　添加 POU _ ST 到任务配置的 MAST

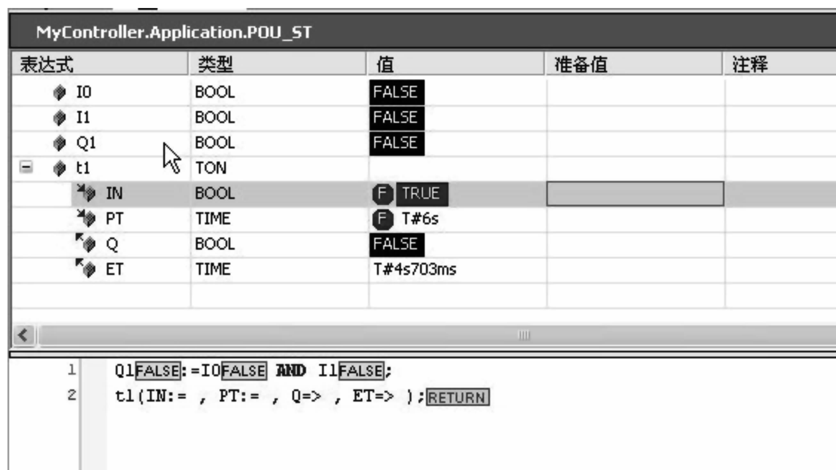


图 5-45　仿真开始运行

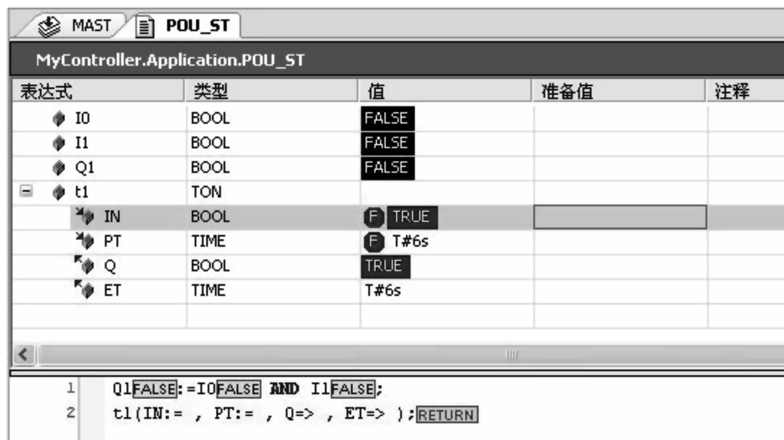


图 5-46　延时结束，Q 变为“真”

格式 2：多个条件的串行

IF 表达式 1 THEN 语句块 1；	如果表达式 1 为“真”，则执行语句块 1。跳到结束。
ELSIF 表达式 2 THEN 语句块 2；	如果表达式 1 为“假”，则再判断表达式 2，表达式 2 为“真”，则执行语句块 2。跳到结束。
END _ IF	

```
(* IF ...THEN...ELSIF  *)
IF V1 THEN Q1:=TRUE;
ELSIF V2 THEN
      Q2:=TRUE;
      V3:=3;
      V4:=4;
ELSIF v5 THEN
    d3:=TRUE;
    v6:=6;
    END_IF
```

格式 3：条件的并行分支

IF 表达式 1 THEN 语句块 1；	如果表达式 1 为“真”，则执行语句块 1。跳到结束。
ELSIF 表达式 2 THEN 语句块 2；	如果表达式 1 为“假”，则再判断表达式 2，表达式 2 为“真”，则执行语句块 2。跳到结束。
ELSE 语句块 3；	如果以上表达式都为“假”，则执行语句块 3。
END _ IF	结束 IF 语句

当前面 v5 为真时，不执行 ELSE 后面的语句。

```
(* IF ...THEN...ELSIF  *)
IF V1[FALSE] THEN Q1[FALSE]:=TRUE;
ELSIF V2[FALSE] THEN
      Q2[FALSE]:=TRUE;
      V3[ 0 ]:=3;
      V4[ 0 ]:=4;
ELSIF v5[TRUE] THEN
    d3[TRUE]:=TRUE;
    v6[ 6 ]:=6;
ELSE
v7[ 0 ]:=7;
v8[ 0 ]:=8;

    END_IF
```

当前面表达式都为假时，执行 ELSE 后面的语句。

```
(* IF ...THEN...ELSIF  *)
IF V1[FALSE] THEN Q1[FALSE]:=TRUE;
ELSIF V2[FALSE] THEN
      Q2[FALSE]:=TRUE;
```

```
        V3 0 :=3;
        V4 0 :=4;
ELSIF v5 FALSE THEN
      d3 TRUE :=TRUE;
      v6 6 :=6;
ELSE
v7 7 :=7;
v8 8 :=8;

      END_IF

```

例 3：多重选择 CASE 表达式 OF…　END _ CASE

CASE 表达式 OF　　写出表达式

值 1：语句 1；　　表达式值为值 1，执行语句 1；

值 2：语句 2；　　表达式值为值 2，执行语句 2；

……

值 n：语句 n；　　表达式值为值 n，执行语句 n；

END _ CASE　　结束 CASE 语句

这种选择语句结构，CASE 的表达式相当于一个指针，指向被执行的语句。

程序事例：

```
(*case of   end_case *)
IF k1 THEN page:=10;
    ELSIF k2 THEN page:=20;
    ELSIF k3 THEN page:=30;
END_IF
CASE page OF
10: s1:=10;
20: s2:=20;
30: s3:=30;
END_CASE

```

仿真情况如下：

```
(*case of   end_case *)
IF k1 TRUE THEN page 10 :=10;
    ELSIF k2 FALSE THEN page 10 :=20;
    ELSIF k3 FALSE THEN page 10 :=30;
END_IF
CASE page 10 OF
10: s1 10 :=10;
20: s2 0 :=20;
30: s3 0 :=30;
END_CASE

```

例 4：语句的循环执行

格式 1：FOR 循环

规则：INT _ VAR：INT；

FOR < INT _ VAR >： =< 初值 > TO < 结束值 > ｛BY < 步长 >｝ DO

语句块；

END _ FOR

这里｛｝内的命令是可选的。语句块的循环执行，一直到〈INT _ VAR〉大于结束值为止。

语句块执行一次，INT _ VAR 值增加一个步长。

参考实例如下：

```
(* for loop *)
s4:=1;
FOR c1:=1 TO 5 BY 1 DO
    s4:=s4*2;
END_FOR
sum:=s4;
```

仿真结果：

```
(* for loop *)
s4[ 32 ]:=1;
FOR c1[ 6 ]:=1 TO 5 BY 1 DO
    s4[ 32 ]:=s4[ 32 ]*2;
END_FOR
sum[ 32 ]:=s4[ 32 ];

```

格式 2：WHILE 循环

规则：WHILE 表达式 DO

　　语句块；

　　END _ WHILE

语句块循环执行，一直到表达式为‘假’为止。

程序案例：

```
(* while loop *)
s5:=1;
c2:=5;
WHILE c2<>0 DO;
    c2:=c2-1;
    s5:=s5*2;

END_WHILE
sum2:=s5;
```

仿真结果如下：

```
(* while loop *)
s5 32 :=1;
c2 0 :=5;
WHILE c2 0 <>0 DO;
    c2 0 :=c2 0 -1;
    s5 32 :=s5 32 *2;

END_WHILE
sum2 32 :=s5 32 ;
```

格式 3：REPEAT 循环

规则：REPEAT 语句块；

UNTIL 表达式

END _ REPEAT

语句块循环执行，直到表达式为“真”停止。

注意规则：表达式后没有分号。

程序案例如下：

```
(* repeat loop *)
s5:=1;
c2:=5;
REPEAT
    s5:=s5*2;
    c2:=c2-1;
    UNTIL
    c2=0
END_REPEAT
sum2:=s5;
```

仿真执行结果：

```
(* repeat loop *)
s5 32 :=1;
c2 0 :=5;
REPEAT
    s5 32 :=s5 32 *2;
    c2 0 :=c2 0 -1;
    UNTIL
    c2 0 =0
END_REPEAT
sum2 32 :=s5 32 ;
```

5.5.2　POU IL 指令表编程方式

指令表编程是组织了一系列指令，并且每条指令包含了操作符和操作数。当有多个操作数时，彼此用逗号分开。在指令前边，可以放置标志码（题标），后跟冒号“:”。这样，执

行语句在跳转时，可以直接找到题标行的指令。

在建立 POU 时，选择实现语言为指令表（IL），如图 5-47 所示。建立的 POU 为 POU _ IL，如图 5-48 所示。



图 5-47　选择指令表

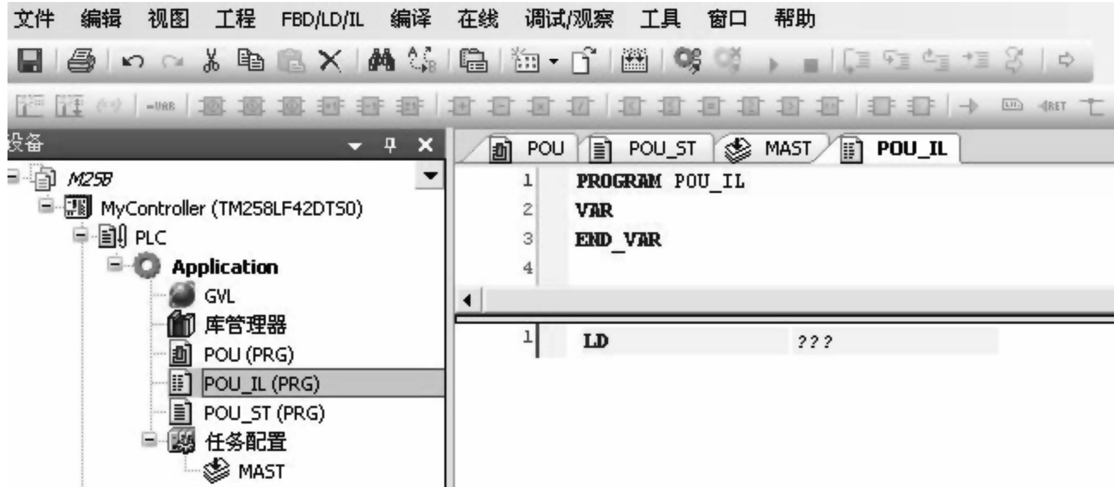


图 5-48　建立 POU _ IL

点击鼠标右键，出现提示栏。选择在下方插入 IL 行。则加入新的指令行，如图 5-49 所示。

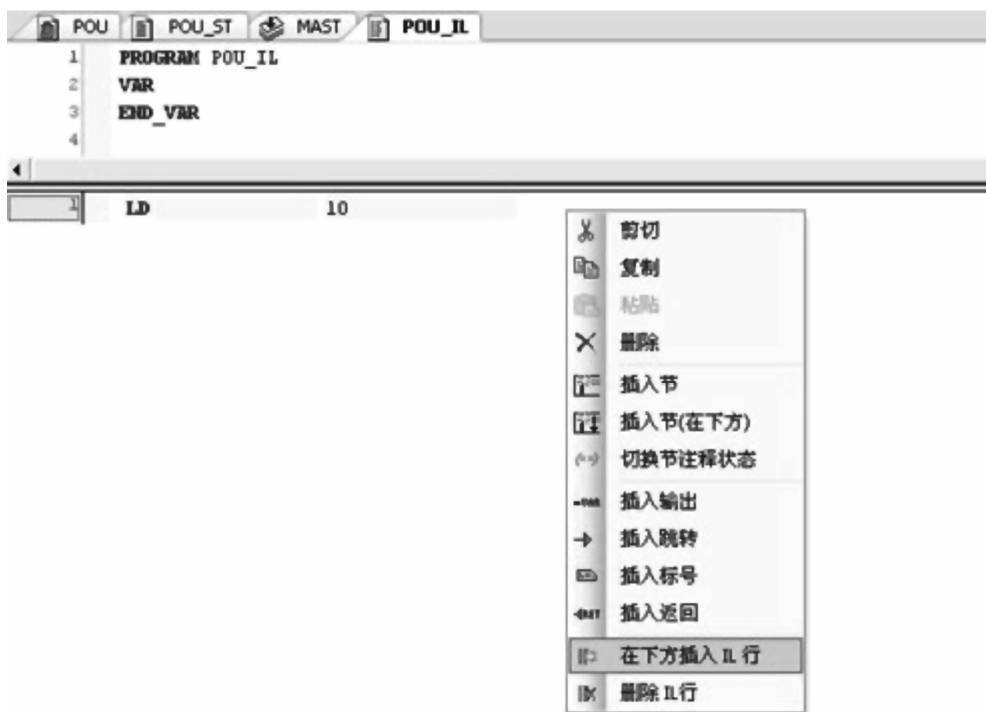


图 5-49　点击鼠标右键，弹出功能列表，选择插入行

指令行由操作符和操作数两部分组成。点击操作符部分，按“F2”，会打开输入助手，如图 5-50 所示。



图 5-50　按“F2”，打开输入助手

在输入助手中有关键字和模块调用。这样就可以很方便地输入指令。例如输入 AND 指令，如图 5-51 所示。

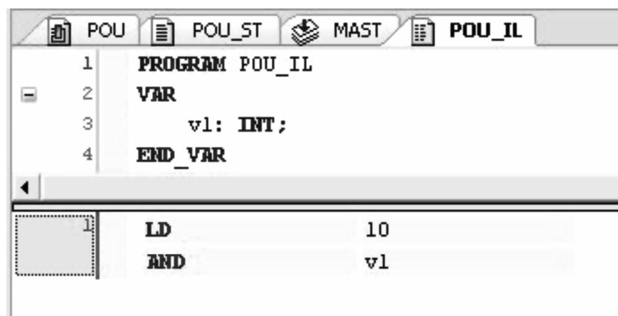


图 5-51　输入指令

再增加一行，写入新指令，如图 5-52 所示。

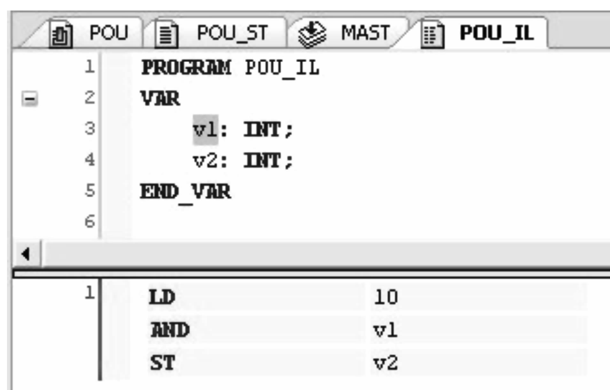


图 5-52　写入新指令

从写程序可以看出 IL 是一种面向行的语言，有点类似于汇编语言。一条指令，是 PLC 可执行的一项命令，它严格要求由一个行来表述。也允许空白行形式的空指令。标准汇编器通常建立在一个微处理器的实际累加器上，即一个值被装入累加器，然后加上，减去其他数值，累加器的最后数值可能存储在一个存储器空间中。而 IL 也提供了一个称为“当前结果”（CR）的累加器。但 CR 并不像实际的累加器那样具有固定数量的存储位。IL 编译程序确保总能提供一个任意存储宽度的虚拟累加器（包括累加器堆栈）。存储位的数量取决于正在处理的操作数的数据类型。与 CR 相关联的数据类型也可变化，以匹配最新的操作数的数据类型。与汇编程序明显不同的是，IL 没有特定的处理器状态位。一个比较操作的判断在 CR 中生成布尔数“真”或“假”。随后的条件跳转或调用，使用 CR 的内容真或假作为执行条件跳转或调用。

一个案例程序及其变量的定义如图 5-53 所示。

程序仿真结果如图 5-54 所示。

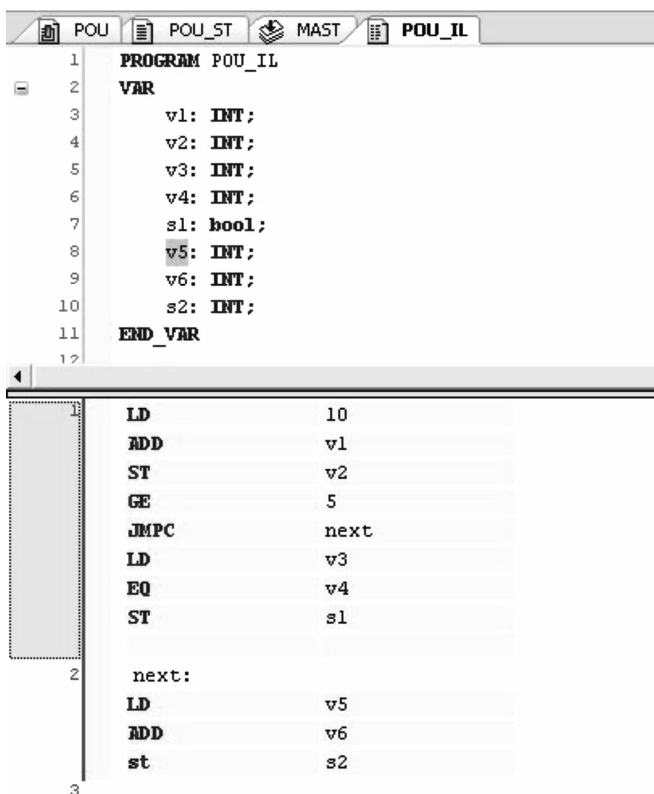


图 5-53　IL 程序案例

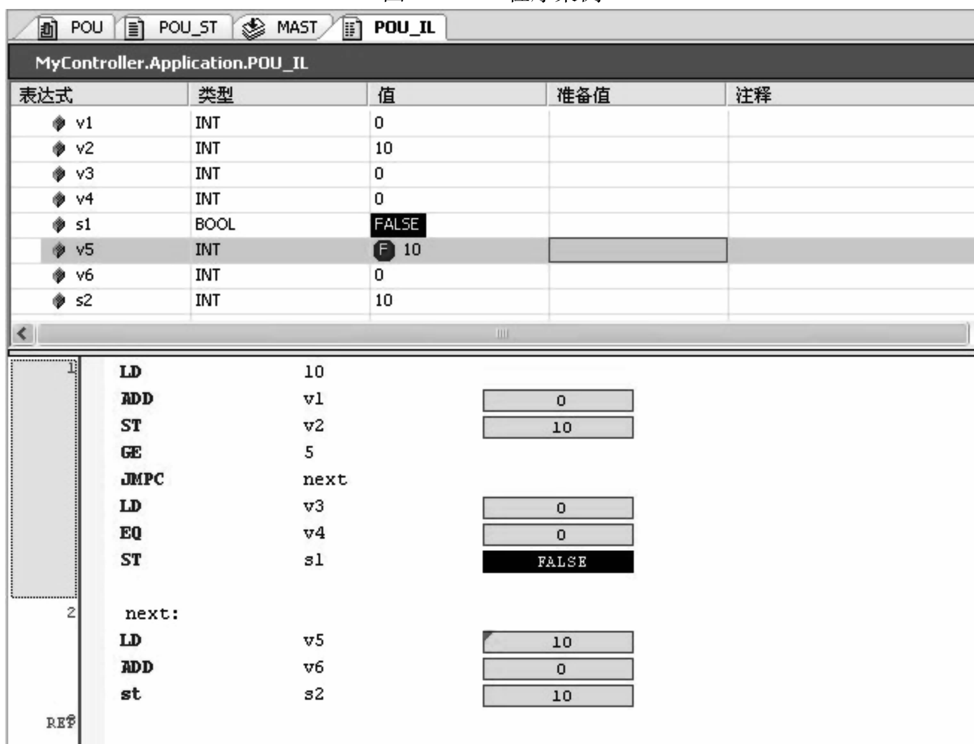


图 5-54　仿真案例程序

5.5.3　POU LD 梯形图编程方式

梯形图编程方式是一种基于继电器逻辑结构搭建的编程方式。它非常直观地描述了程序从左到右，从上到下的执行顺序。非常适合于逻辑运算的编辑。

在建立 POU 时，选择实现语言为梯形逻辑图（LD），如图 5-55 所示，建立的 POU 为 POU _ LD。



图 5-55　创建 POU _ LD

点击打开，如图 5-56 所示。



图 5-56　打开 POU 的梯形图编程

用鼠标点击编程行，编程区上方会出现梯形图编程的各种元素。或者点击鼠标右键，会出现下拉菜单，菜单中列出了编辑程序所需的各种功能，如图 5-57 所示。

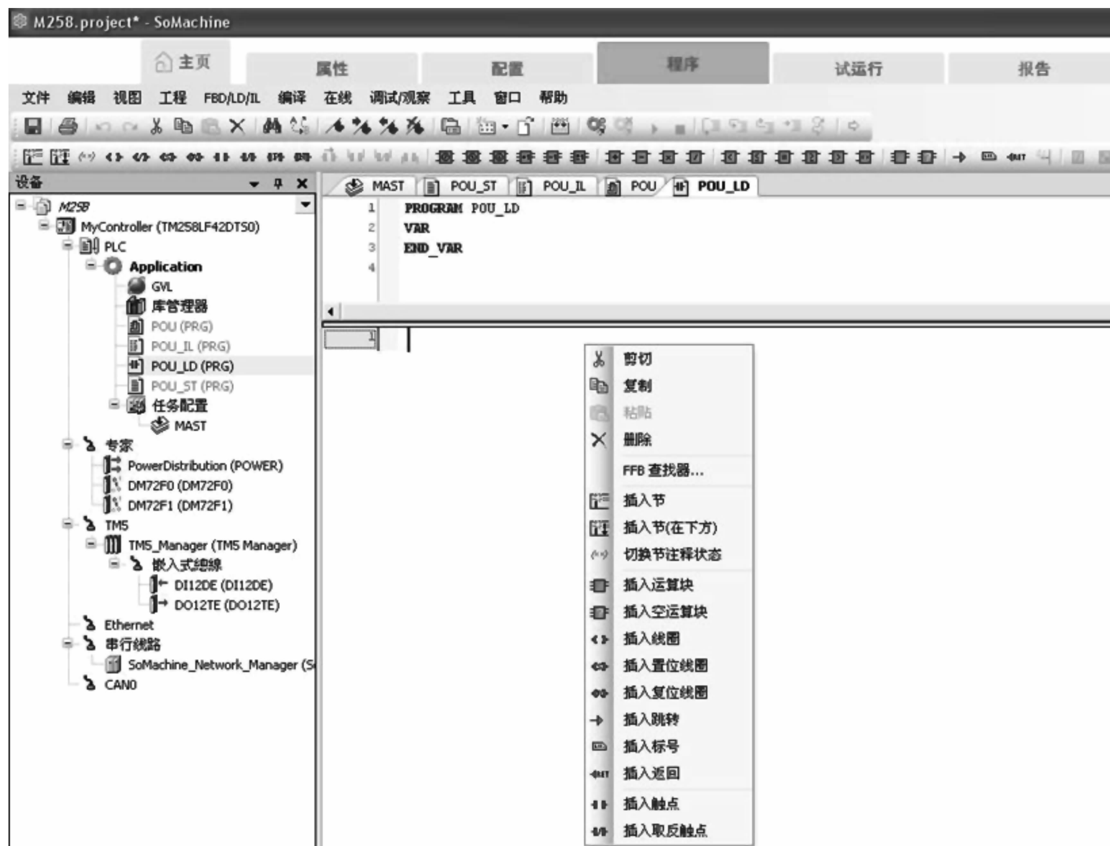


图 5-57　打开编辑菜单

插入一个触点，定义一个变量或点击问号边上的图标，打开变量表，选择一个已经定义好的变量，如图 5-58 所示。

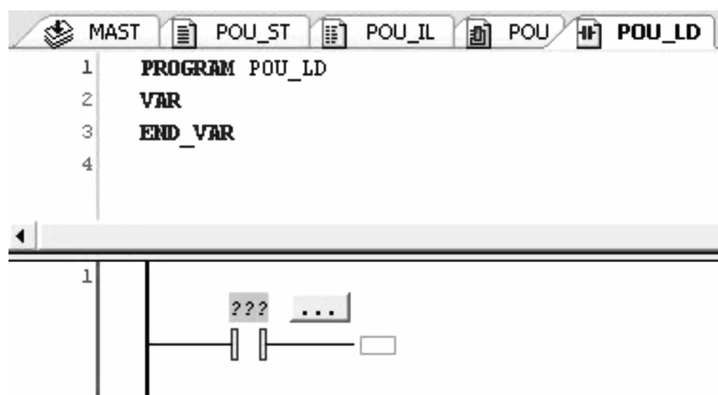


图 5-58　加入一个触点

在这里，我们定义一个变量 K1。写入 K1 并按回车，这时会自动弹出变量定义声明。选择变量类型，按“确定”，K1 就会出现在变量定义栏目，如图 5-59 所示。



图 5-59　定义触点变量

完成一个元素的添加如图 5-60 所示。

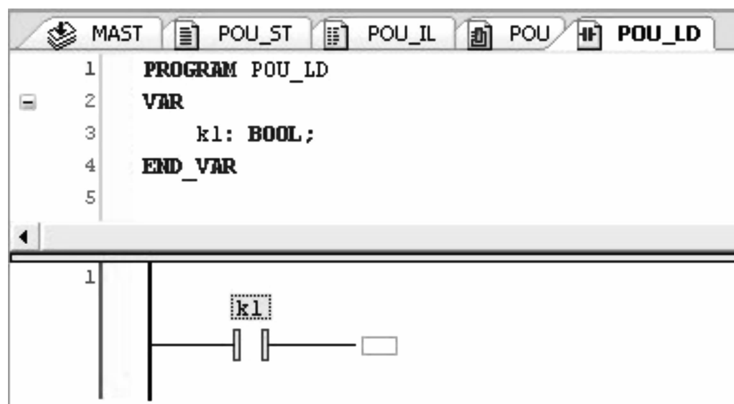


图 5-60　完成一个元素的添加

如此，可以加入串联开关或并联开关，最后加上一个线圈，就完成了一个简单逻辑的编程。

程序的注释可以采用（＊＊）标注，英语直接写入括号内。而汉字需要采用“Ctrl + c”复制，然后用“Ctrl + v”来粘贴到括号内。带有注释的程序如图 5-61 所示。

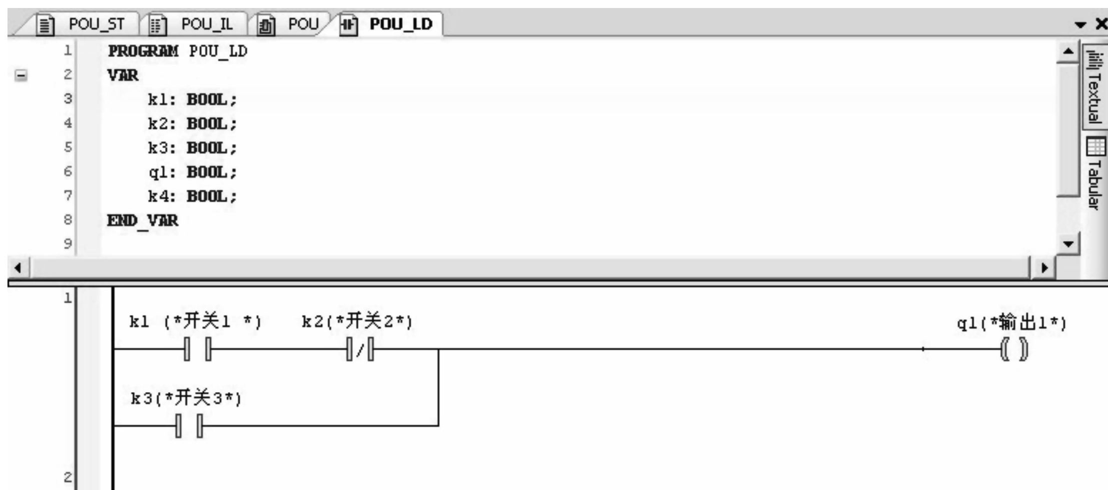


图 5-61　带有注释的程序

仿真程序运行结果如图 5-62 所示。

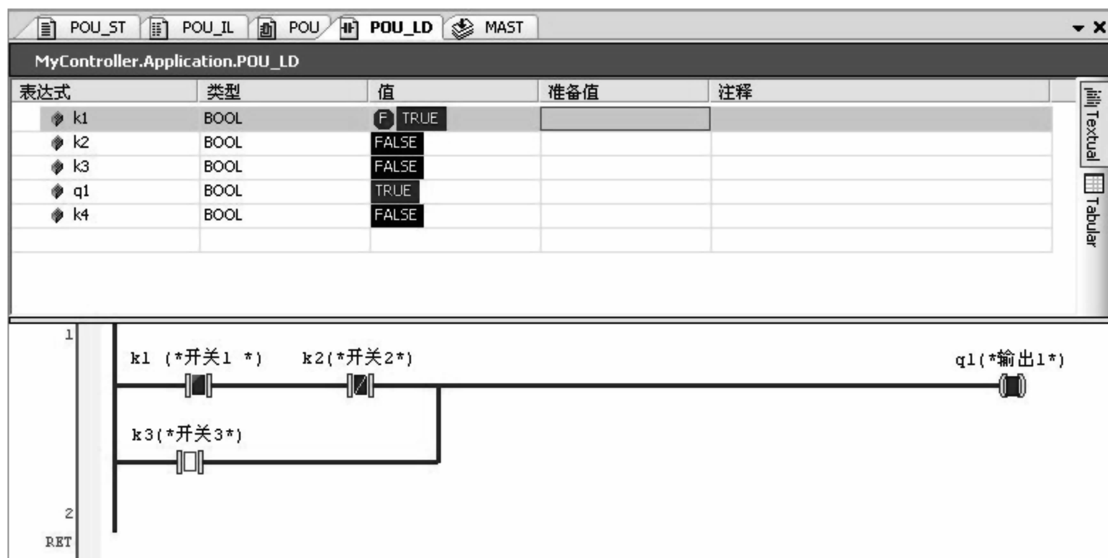


图 5-62　仿真程序运行结果

除了编辑基本元素的逻辑功能外，还能完成各种程序、功能块的调用。按照图 5-57 打开的菜单，点击插入运算块，弹出输入助手，如图 5-63 所示。在这个框架内，有功能块：包含了各种标准的位操作、触发器、计数器、计时器和运算函数、算式及施耐德控制器功能块；有模块调用：包含了各种各种系统命令和其他程序；有关键字：包含了各种操作符；有转换操作符：用于完成各种数据类型的转换。

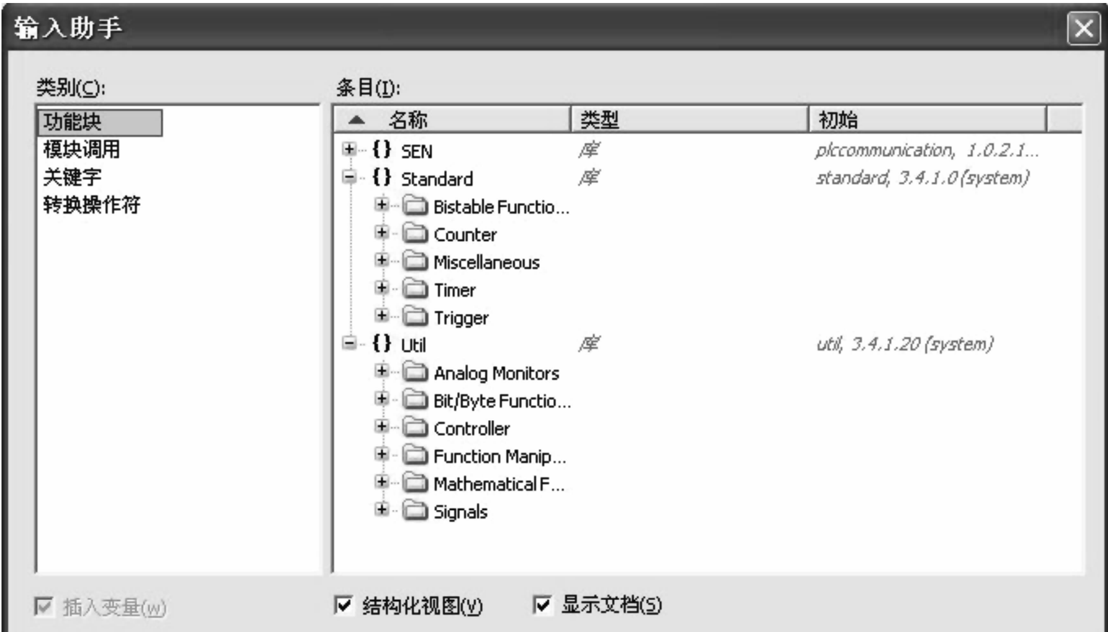


图 5-63　输入助手

借助于输入助手，可以很方便地实现各种功能和运算操作。

例如：建立一个延时功能。

在输入助手框内，选择功能块—Standard—Timer—TON，按“确定”，出现如图 5-64 所示画面。

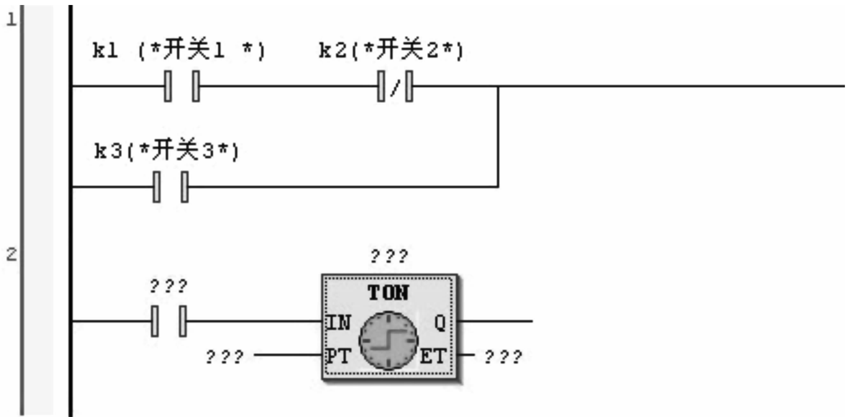


图 5-64　添加一个定时

点击问号，命名功能块名，定义相关参数。定时程序如图 5-65 所示。

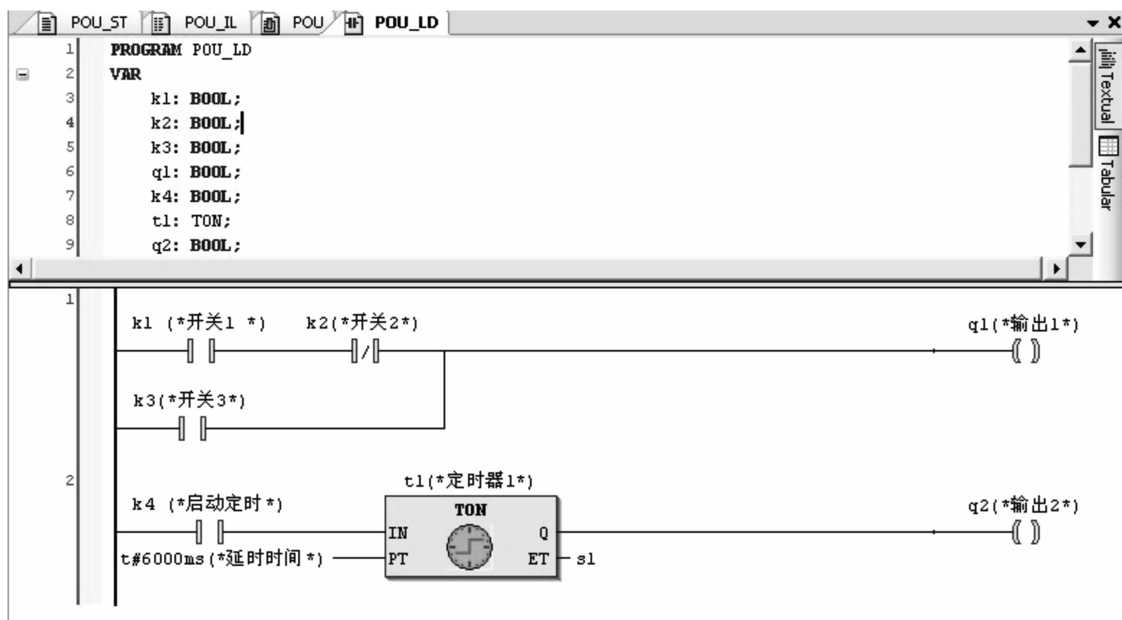


图 5-65　定时程序

仿真情况如图 5-66 所示。

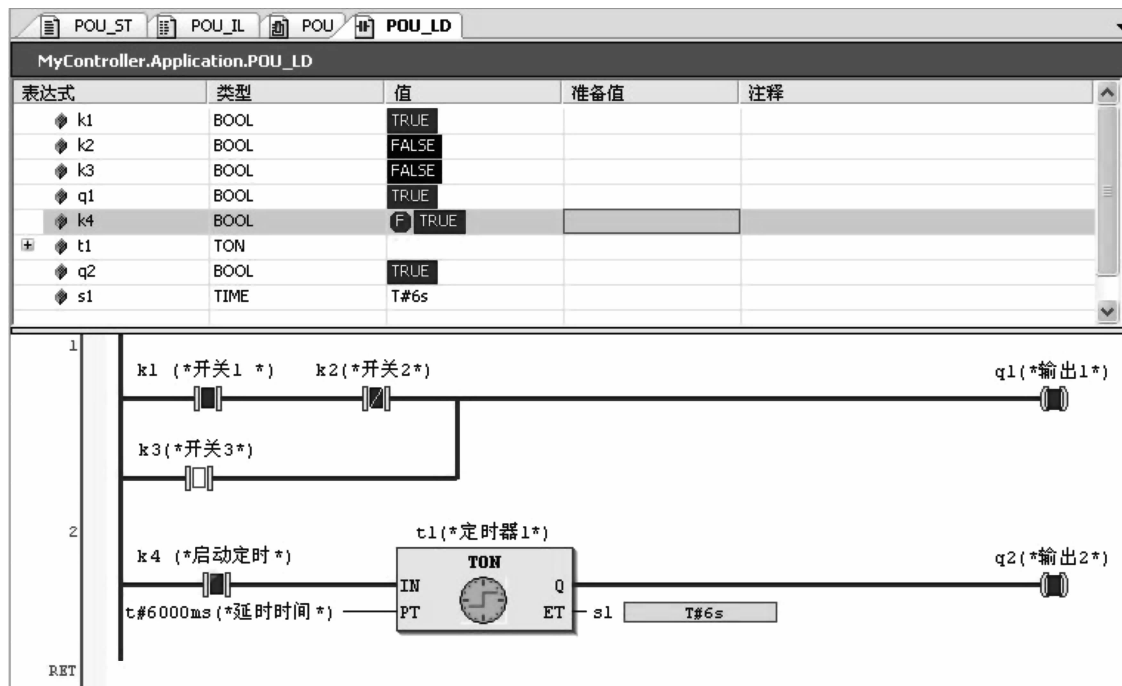


图 5-66　延时程序的仿真

5.5.4　POU FBD 功能块图编程方式

FBD 功能块编程方式是一种功能块流程图方式的图形编程语言。整个程序随着一个个

功能块的搭建，顺序执行，流向清晰，功能明显。它的程序结构由一段一段的结构区组成。使整个程序的实现呈现出清晰的步序。在建立 POU 时，选择编程语言为功能块图（FBD），建立 POU _ FBD，如图 5-67 所示。



图 5-67　建立 POU _ FBD

点击编程行，并点击鼠标右键会出现下拉菜单。我们可以通过下拉菜单输入各种功能块。也可以借助工具箱来找到我们需要的功能，加入到程序中。还可以直接点击编程区上部的功能图标来加入要编辑的功能。例如打开下拉菜单，选择插入运算块，再选择关键字，选择 ADD 功能，按“确定”，功能块便加入到程序行，如图 5-68 所示。

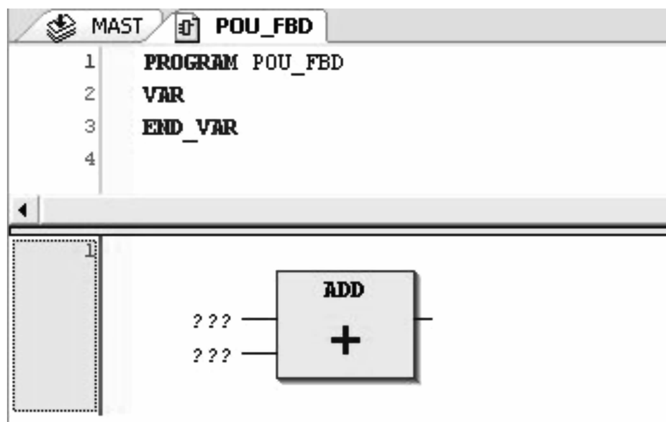


图 5-68　添加功能块 ADD

在输入的问号处定义输入变量，如 s1，定义数据类型并确定，如图 5-69 所示。

同样，再定义第二个输入 s2，完成功能块输入的定义，如图 5-70 所示。

如果功能块要求更多的输入，则鼠标点在功能块内，按鼠标右键，会出现下拉菜单，选择插入输入，如图 5-71 所示。



图 5-69　定义输入变量

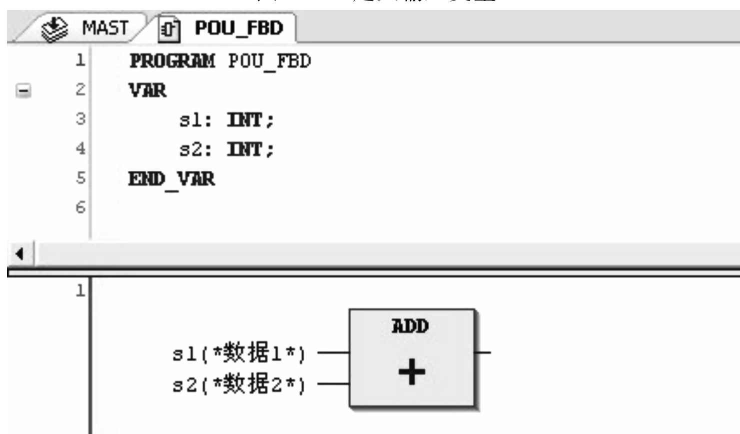


图 5-70　完成两个输入定义

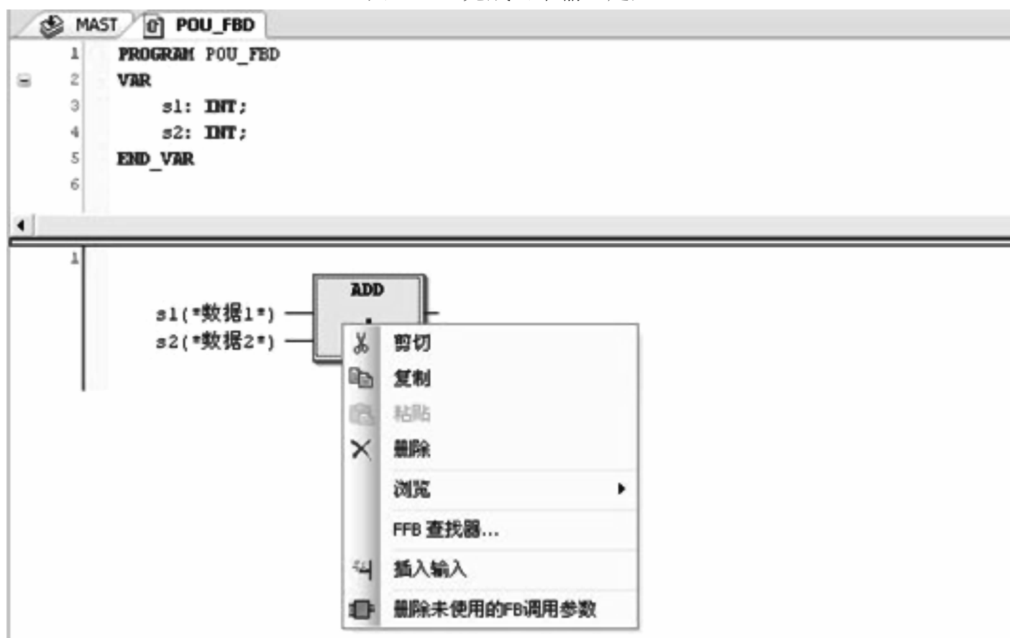


图 5-71　增加一个输入

完成增加了一个输入。也可根据要求增加或删除输入，如图 5-72 所示。

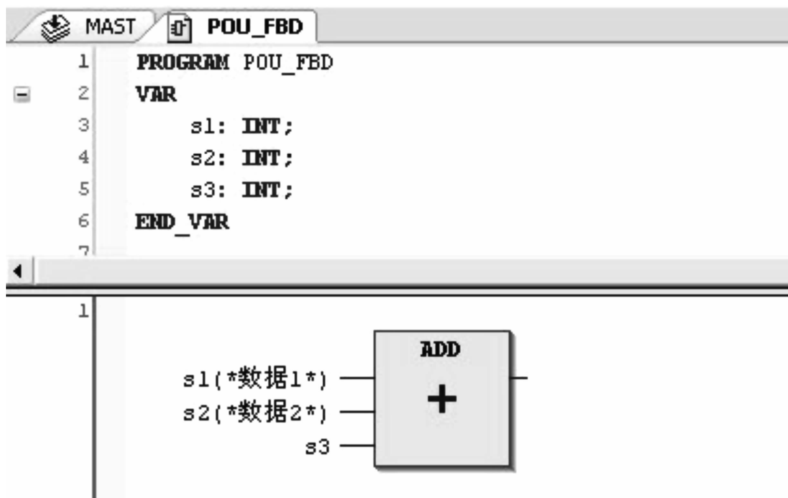


图 5-72　增加输入的定义

在定义输出变量时，鼠标点在编程行功能块外部空间并按右键，打开含有插入输出的菜单，并选择。输出管脚的变量定义如图 5-73 所示。

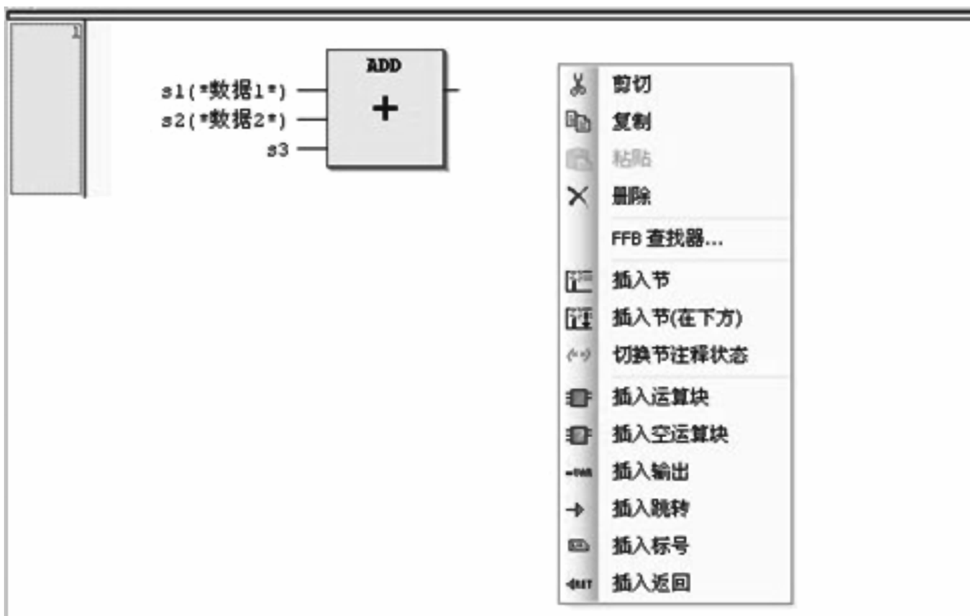


图 5-73　输出管脚的变量定义

添加输出变量如图 5-74 所示。

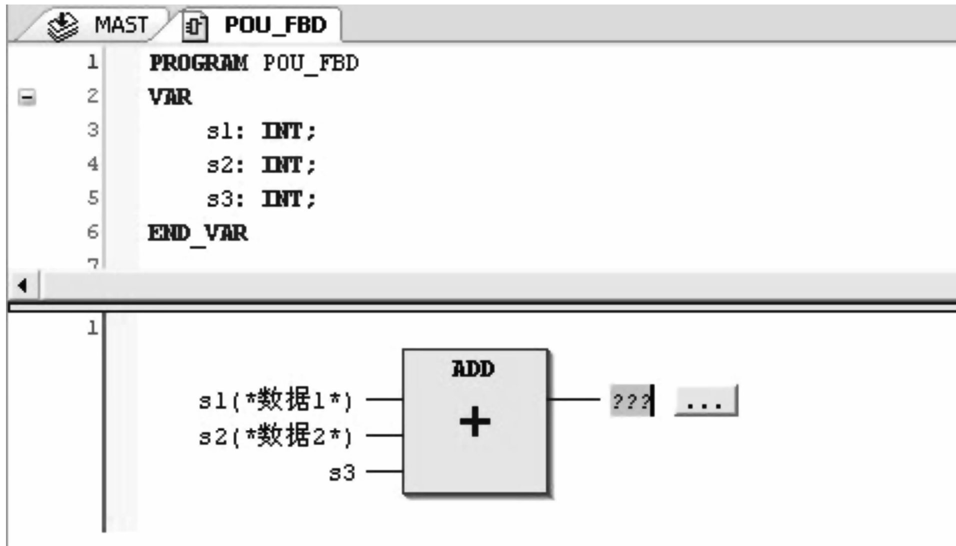


图 5-74　添加输出变量

定义输出变量 result，完成一个加法功能如图 5-75 所示。

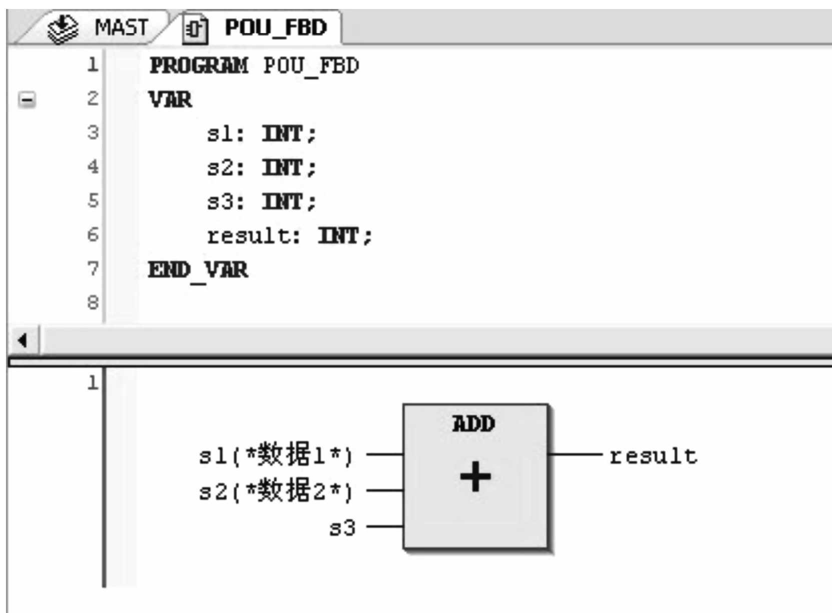


图 5-75　完成一个加法功能

经过仿真可验证程序功能，如图 5-76 所示。

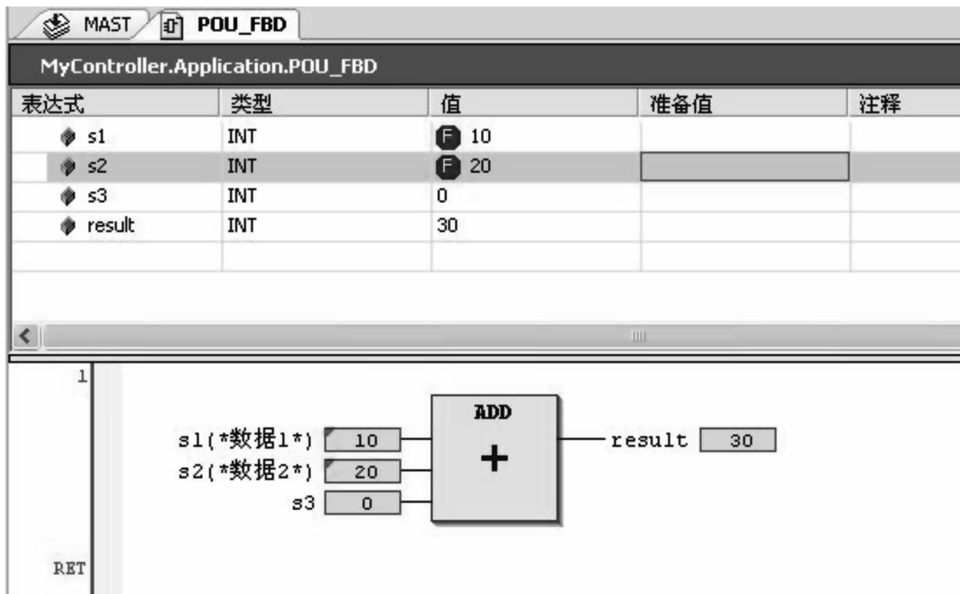


图 5-76　仿真验证功能

5.5.5　POU CFC 连续功能图编程方式

CFC 连续功能图编程类似于功能块图（FBD）方式。但与 FBD 不同的是，它的编程没有编程行的限制，功能块可以放在任何地方。这使得它的编程可以随心所欲。也成为经常使用的一种编程方式。

在建立 POU 时，选择实现语言为连续功能图，如图 5-77 所示。

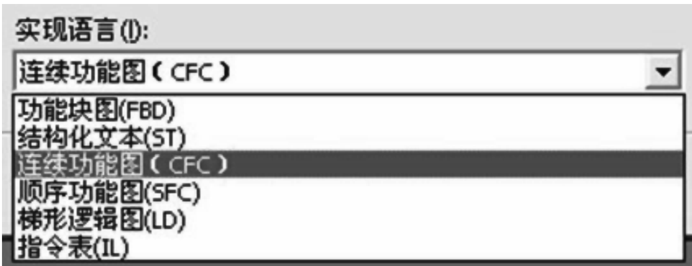


图 5-77　选择连续功能图

确认后，建立 CFC 编程环境，如图 5-78 所示。

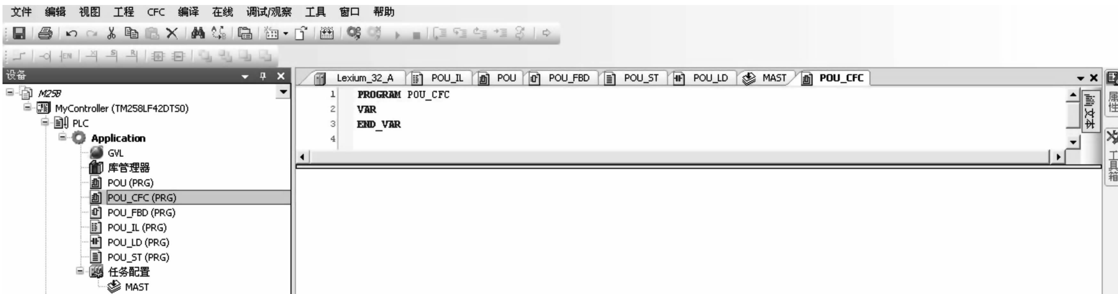


图 5-78　建立连续功能图编程环境

用鼠标点击右侧的工具箱，工具箱下拉菜单打开，如图 5-79 所示，点击运算块，如图 5-80 所示。

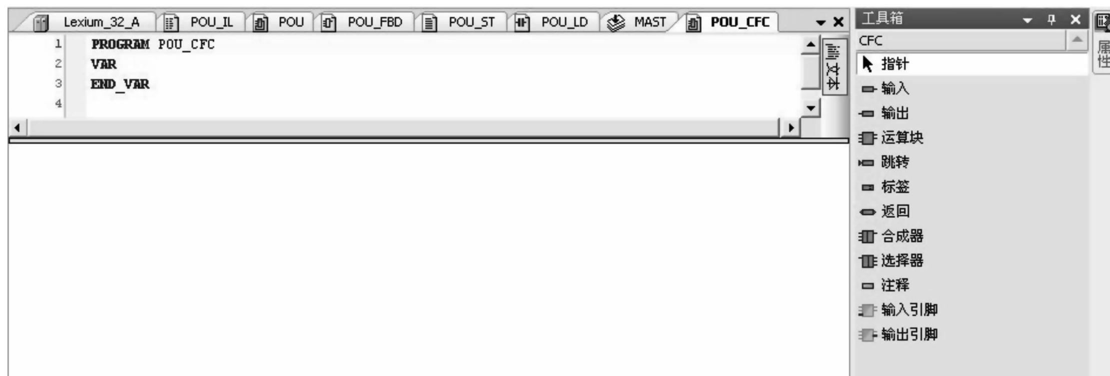


图 5-79　打开工具箱菜单

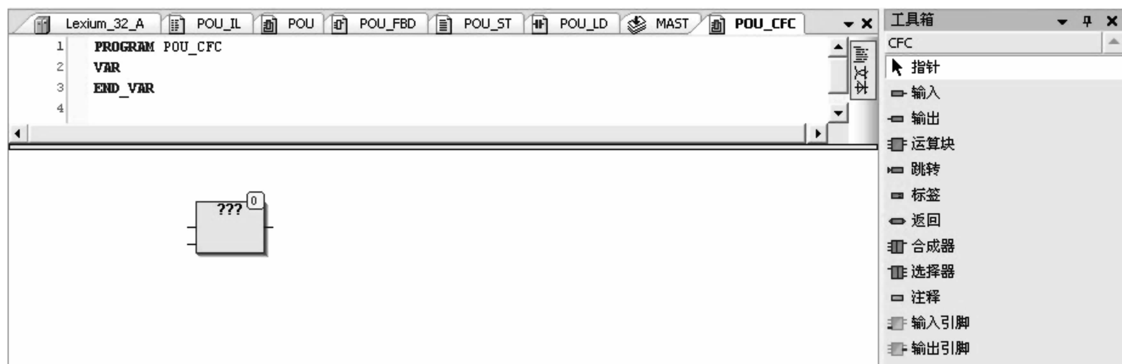


图 5-80　调入一个运算块

点击运算块??? 会出现一个白框，如图 5-81 所示。

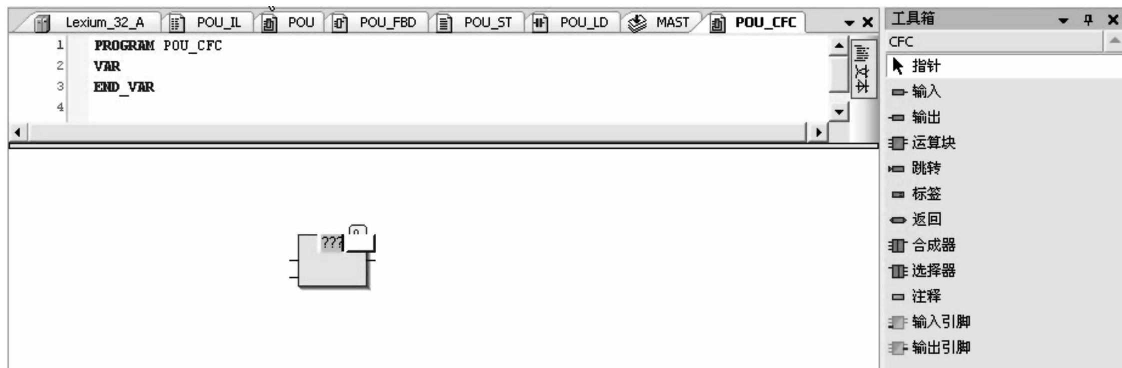


图 5-81　点击运算块，使白框出现

点击白色方框，会出现输入助手，如图 5-82 所示。

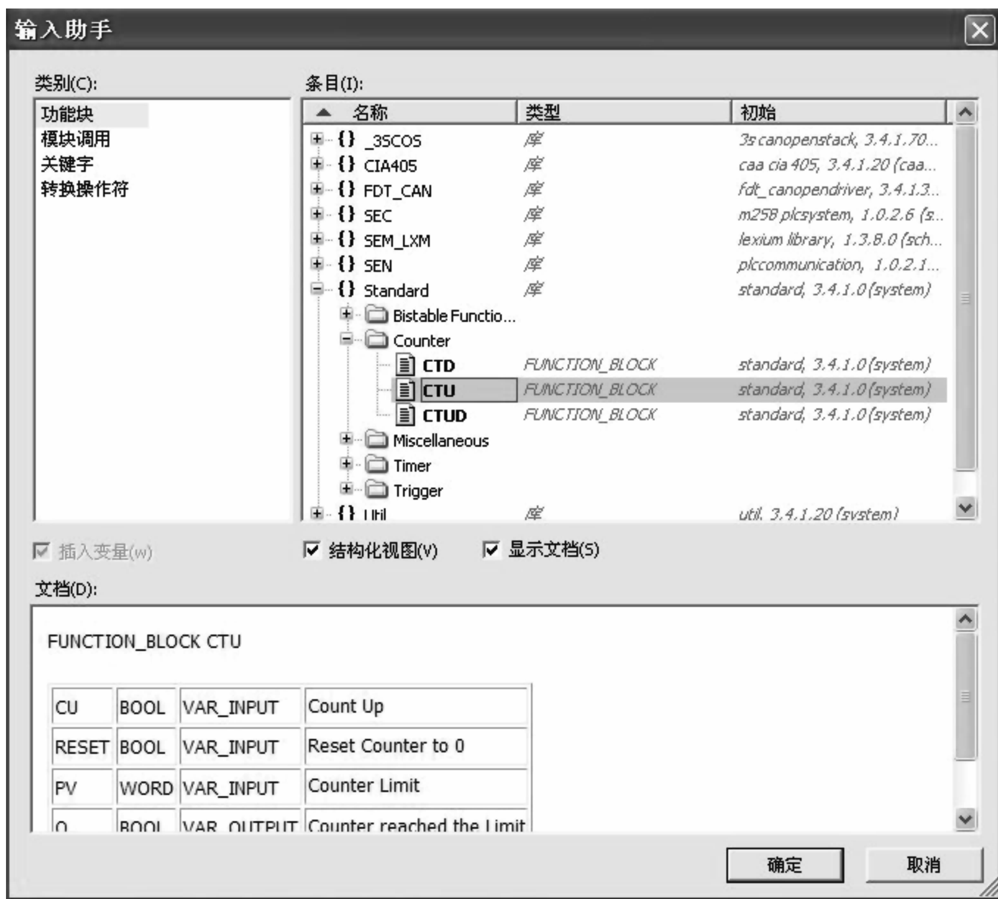


图 5-82　输入助手

选择功能块中的计数模块，点击“确定”，就把功能块加入到编程区域中。

确认运算块功能如图 5-83 所示。

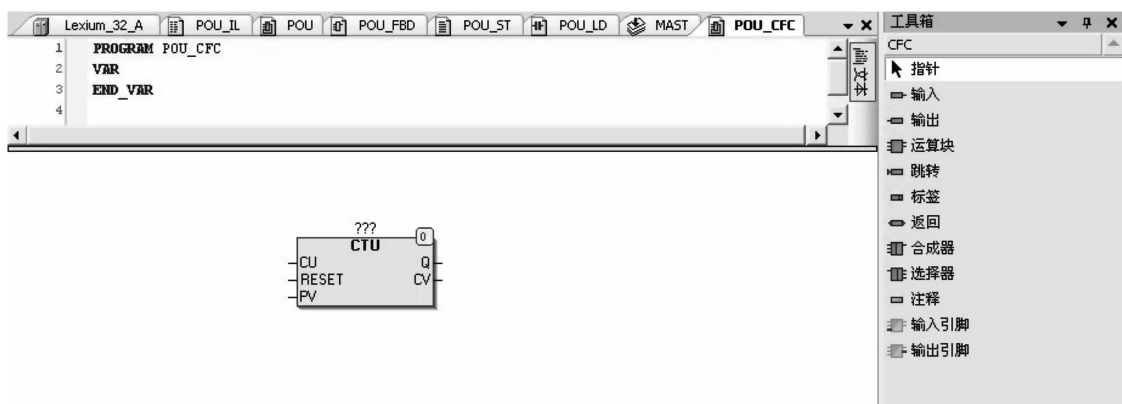


图 5-83　确认运算块功能

给这个功能块命名为 d1 _ ctu 后，会自动弹出对这个名字的解释，如图 5-84 所示。

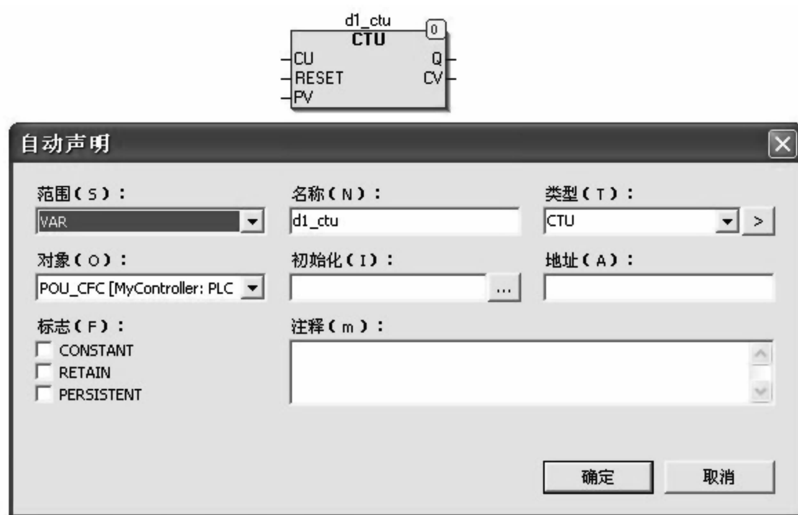


图 5-84　对功能块名字的解释

点击“确定”，则功能块解释进入到变量解释区域，从而完成了一个功能块的调用，如图 5-85 所示。

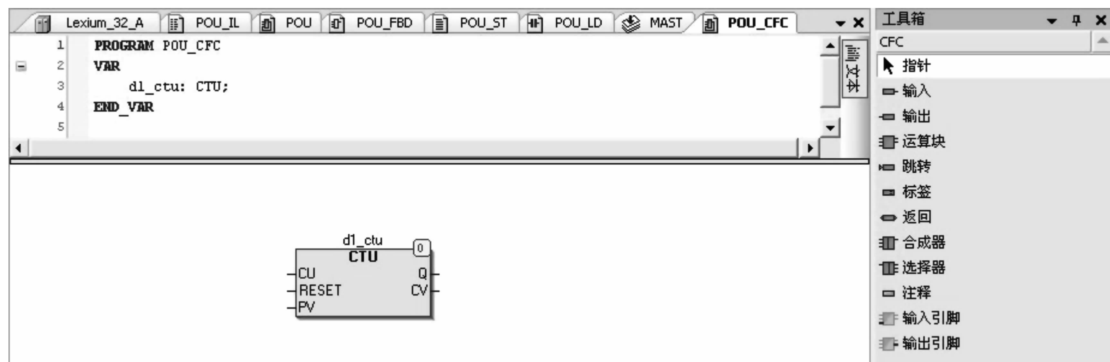


图 5-85　完成功能块调用

点击功能块输入、输出管脚并命名，如图 5-86 所示。

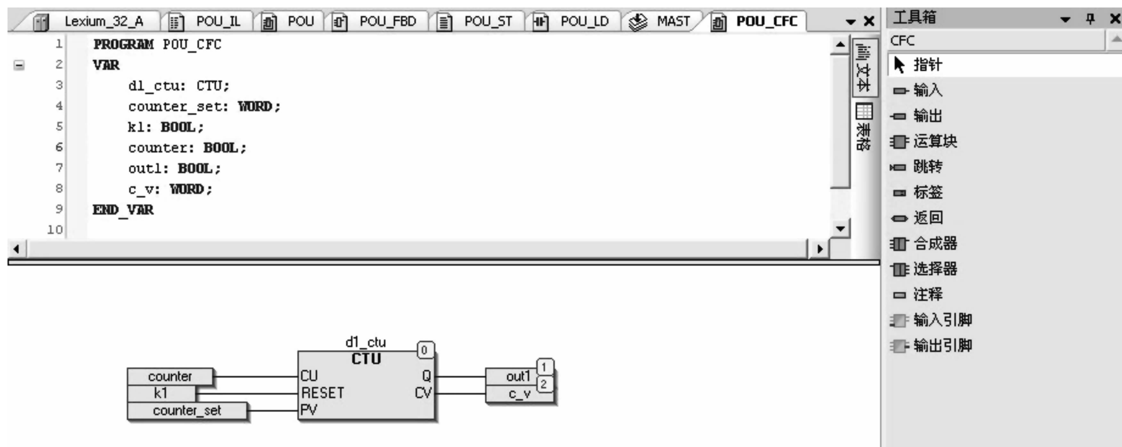


图 5-86　写入输入、输出管脚变量名

这样，我们给 counter _ set 赋值 10，随着 counter 脉冲的输入，计数值 c _ v 增加。计数功能块仿真运行如图 5-87 所示。

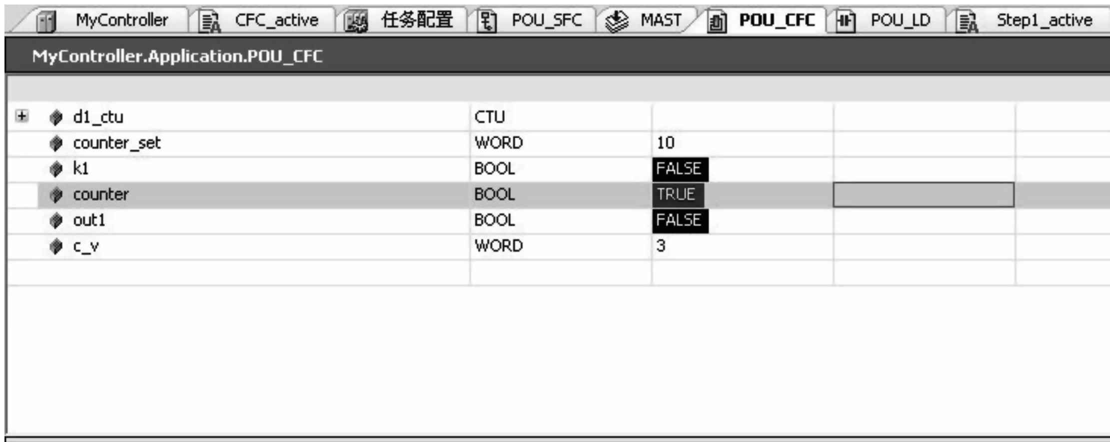




图 5-87　计数功能块仿真运行

当计数值增加到 10，out1 输出，如图 5-88 所示。



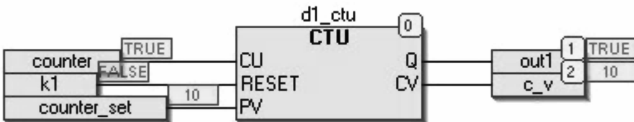


图 5-88　计数到值输出

5.5.6　POU SFC 顺序功能图编程方式

SFC 顺序功能图编程方式是一种图形化编程模式，它的好处就是把很多不同功能的复杂程序分割为较小的可管理的单元，并管理和控制这些单元之间的执行流。这就使得程序的控制结构非常清晰，执行程序的效率得到了提高。

就像指挥交通一样，指挥得好，窄的马路也会畅通。指挥得不好，再宽的马路也会拥堵不堪，行动缓慢。不要指望马路不断扩宽，要不断挖掘有效的管理。编程也是一样，要把不同的程序执行规划好，就能提高效率。SoMachine 平台给出了制定规则的手段，这就是 SFC 编程语言。大家可用这个语言来搭建你的程序结构，优化执行规则。

如果编写的程序条理不清，变量繁杂无定义规则，就像公路交通没有警察和红绿灯，后果是可想而知的，再宽阔的马路也会拥堵不堪，行动缓慢。把所有程序都堆到 MAST 任务，参与扫描，无疑会增加扫描时间，如图 5-89、图 5-90 所示。

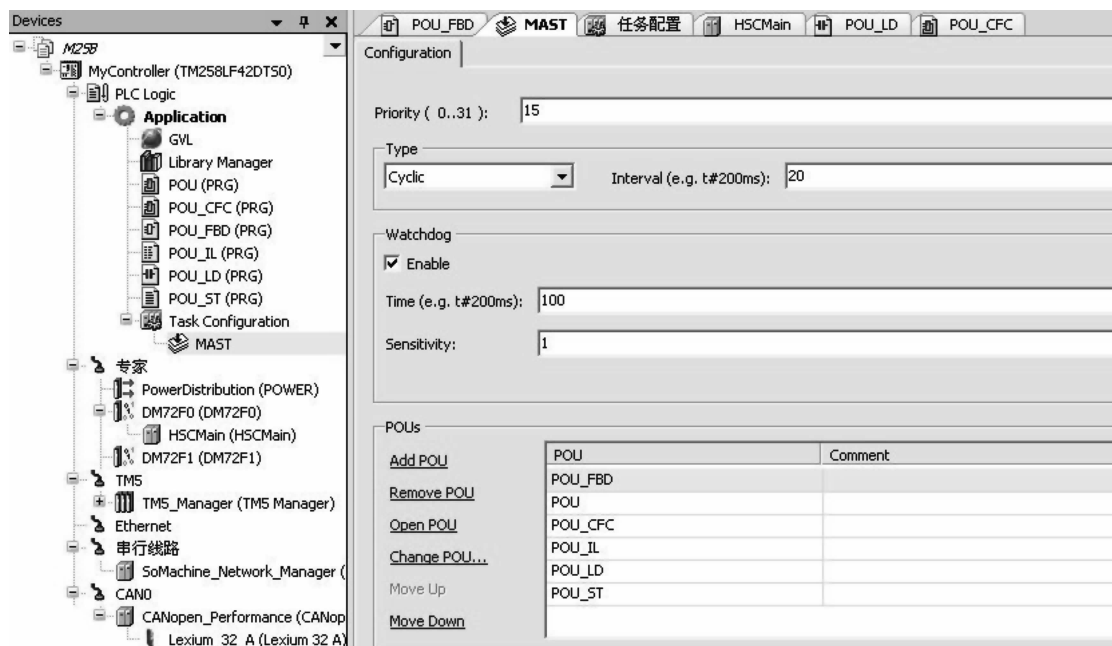


图 5-89　所有程序都堆到 MAST 任务

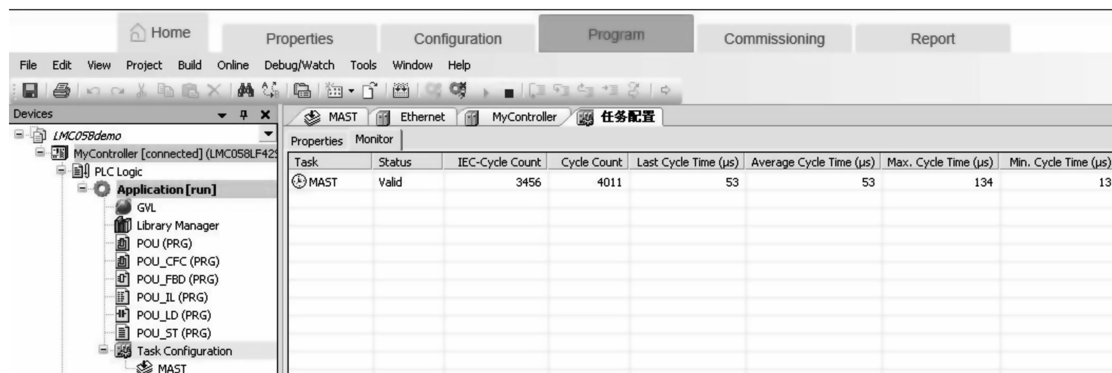


图 5-90　增加了扫描时间

采用 SFC 编程可以合理规划程序结构，让干事儿的程序及时参与扫描，让辅助程序按需加入扫描，从而大大提高 CPU 的使用效率，缩短程序扫描时间，如图 5-91、图 5-92 所示。

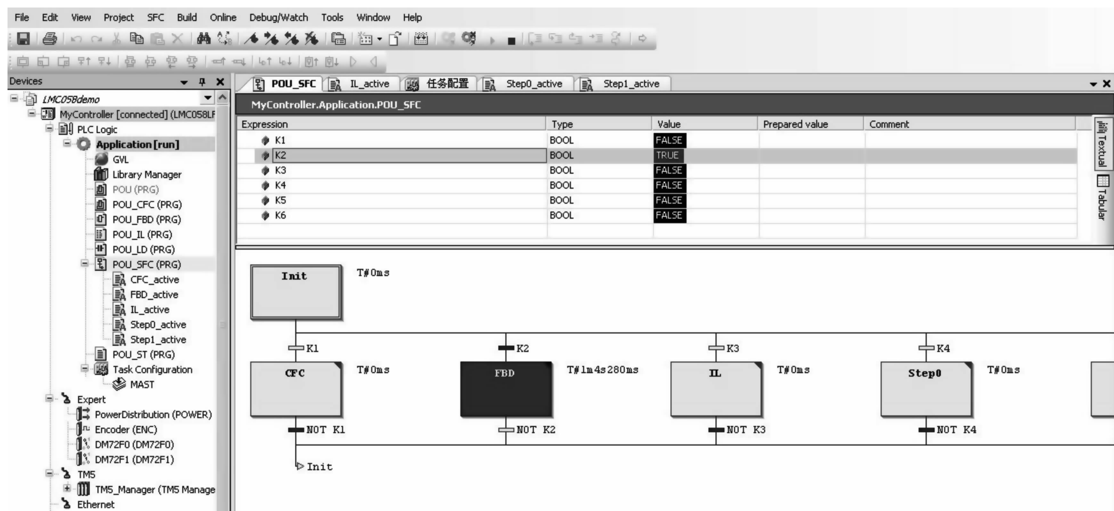


图 5-91　干事儿的程序及时参与扫描

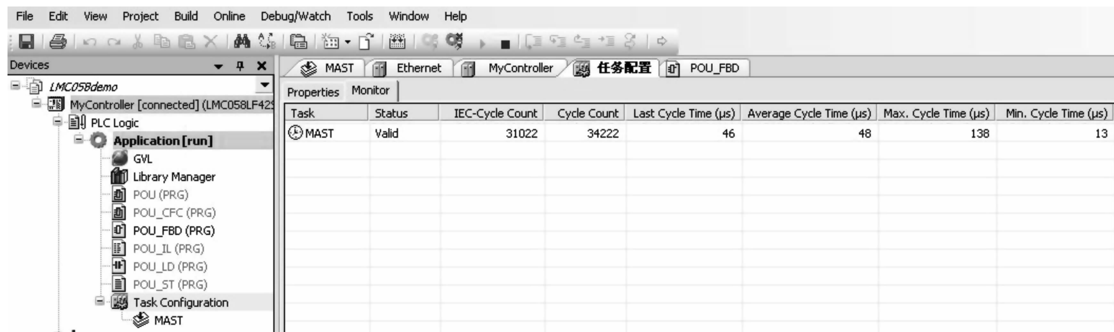


图 5-92　减少了扫描时间

采用 SFC 编程的结构，在创建 POU 程序组织单元时，在实现语言菜单中，选择 SFC 即可，如图 5-93 所示。

点击“打开”后，出现如图 5-94 所示画面。

在编程区域内出现的双线框为初始化功能框，双击“Init”，如图 5-95 所示，可建立一个初始化程序例如 Init _ active，采用结构化文本方式编程，如图 5-96 所示。

在打开的编程区里写入一段程序或一段赋值语句。

自动弹出的变量名解释，填好确认后，都写入 POU _ SFC 主程序的变量声明表里，如图 5-97 所示。

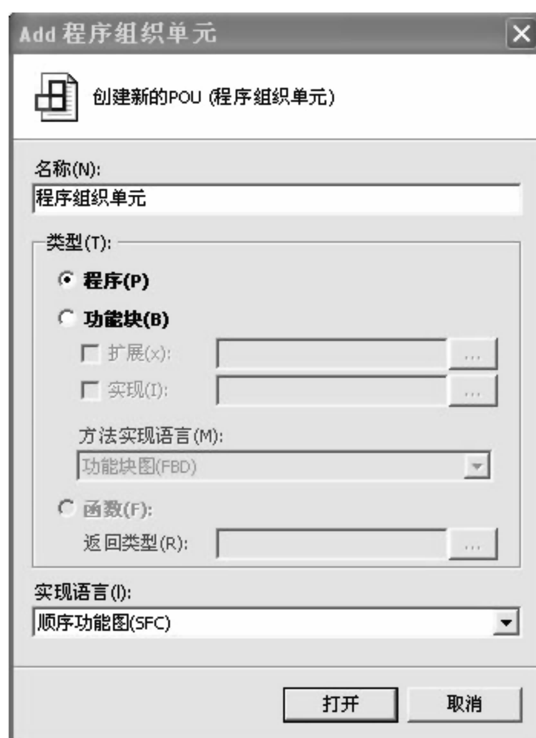


图 5-93　选择 SFC

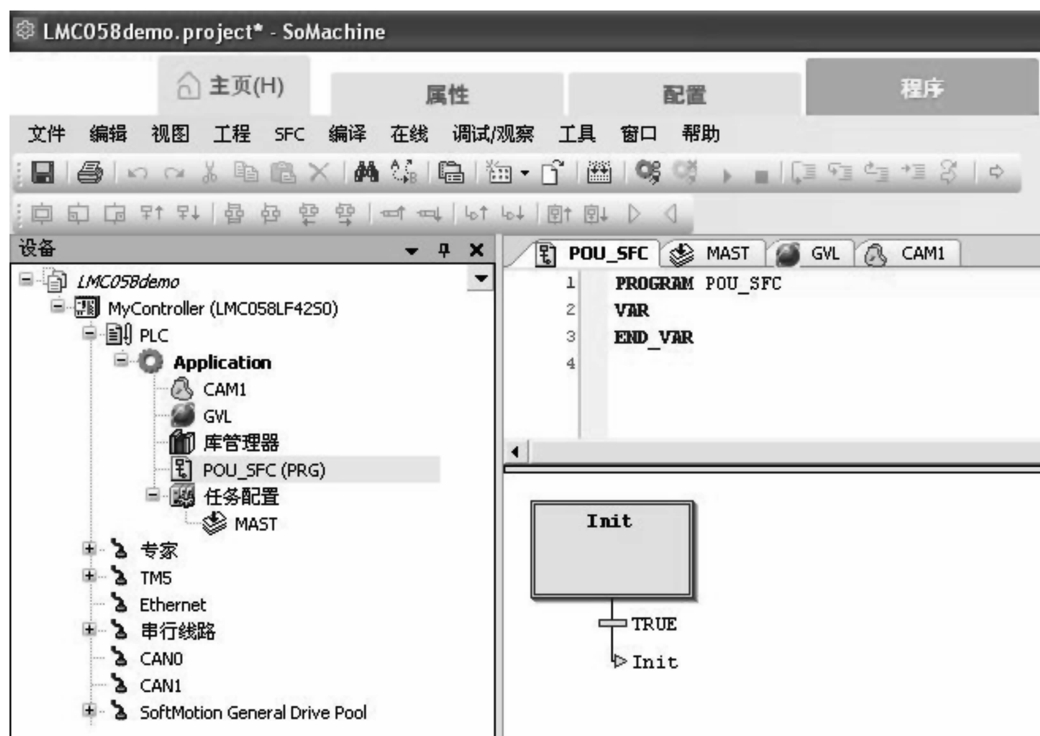


图 5-94　打开 SFC 编程画面



图 5-95　建立功能框内执行程序

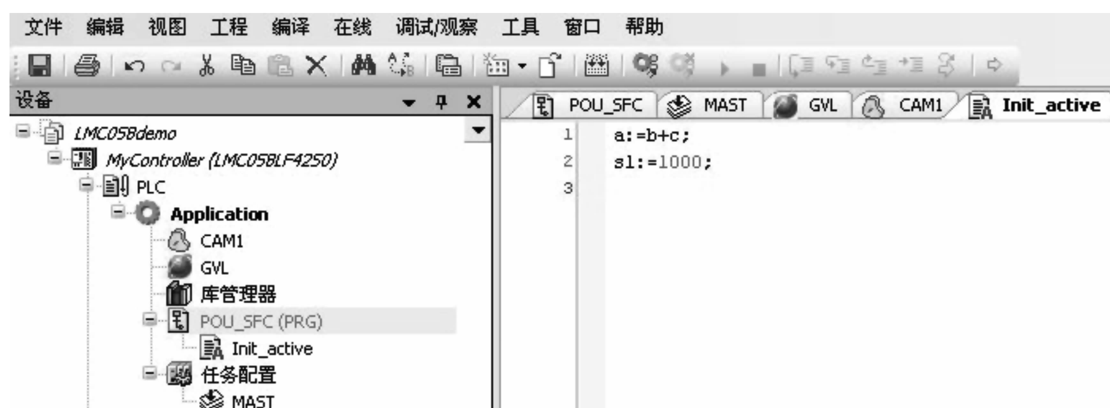


图 5-96　编辑一段执行程序

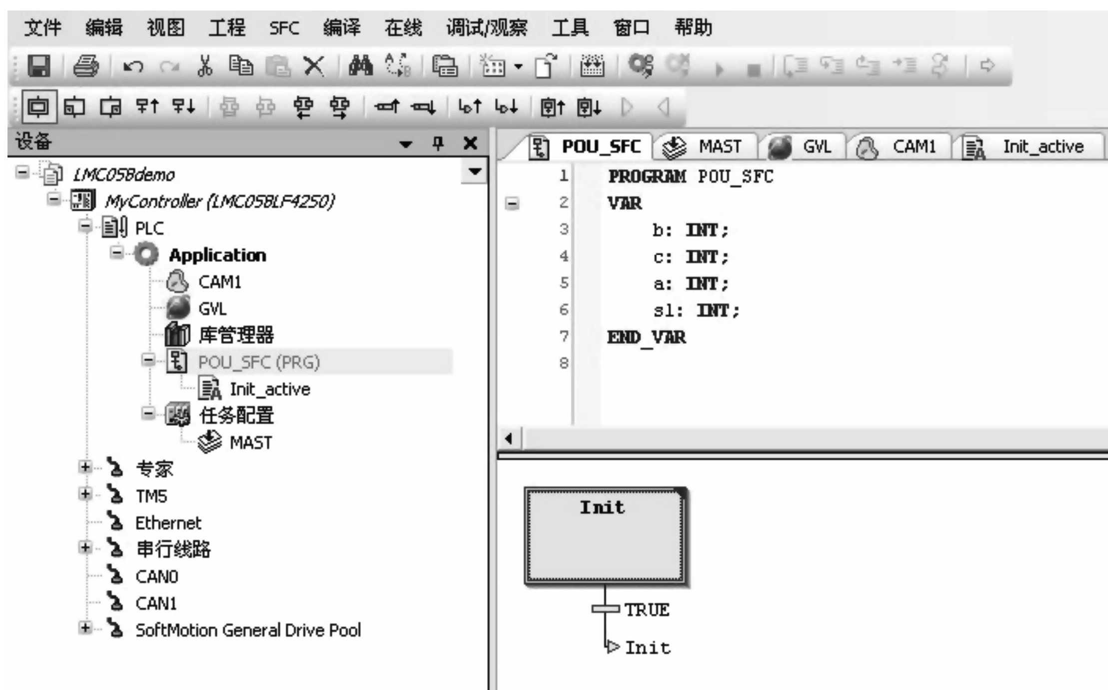


图 5-97　变量声明

我们把 POU _ SFC 配置到任务表里，参与运行扫描。点击任务配置下的 MAST，打开配置如图 5-98 所示。

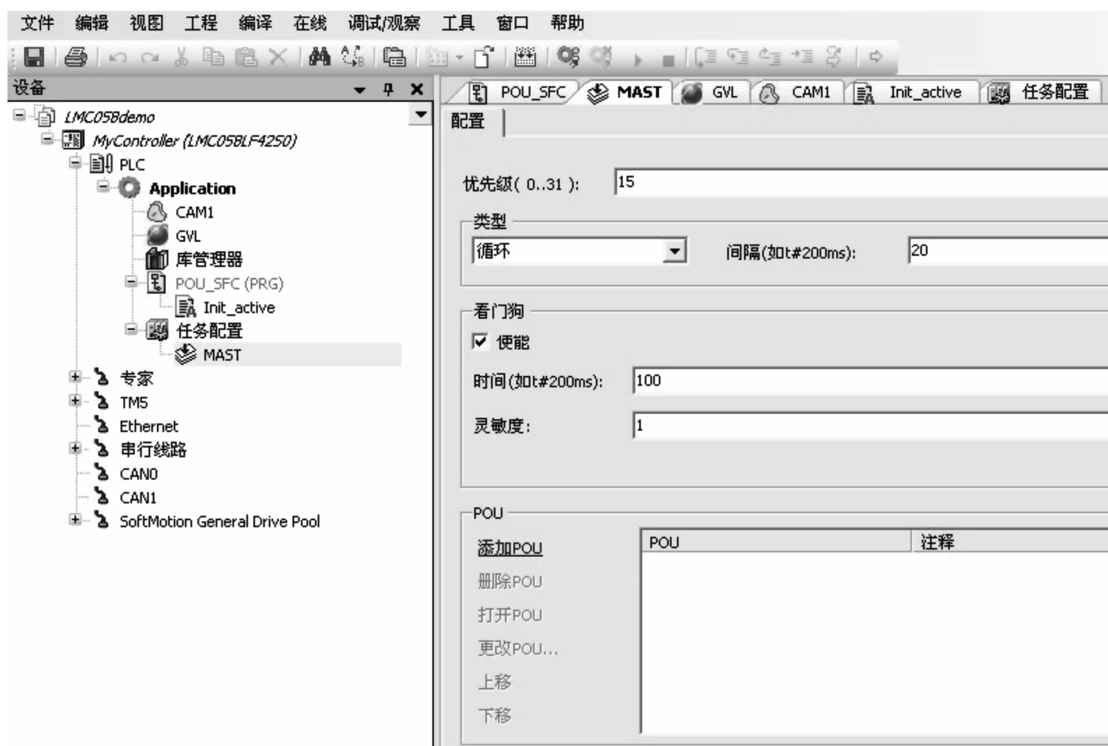


图 5-98　配置执行程序

点击添加 POU。打开输入助手，点击“Application”选择 POU _ SFC，按“确定”，如图 5-99 所示。

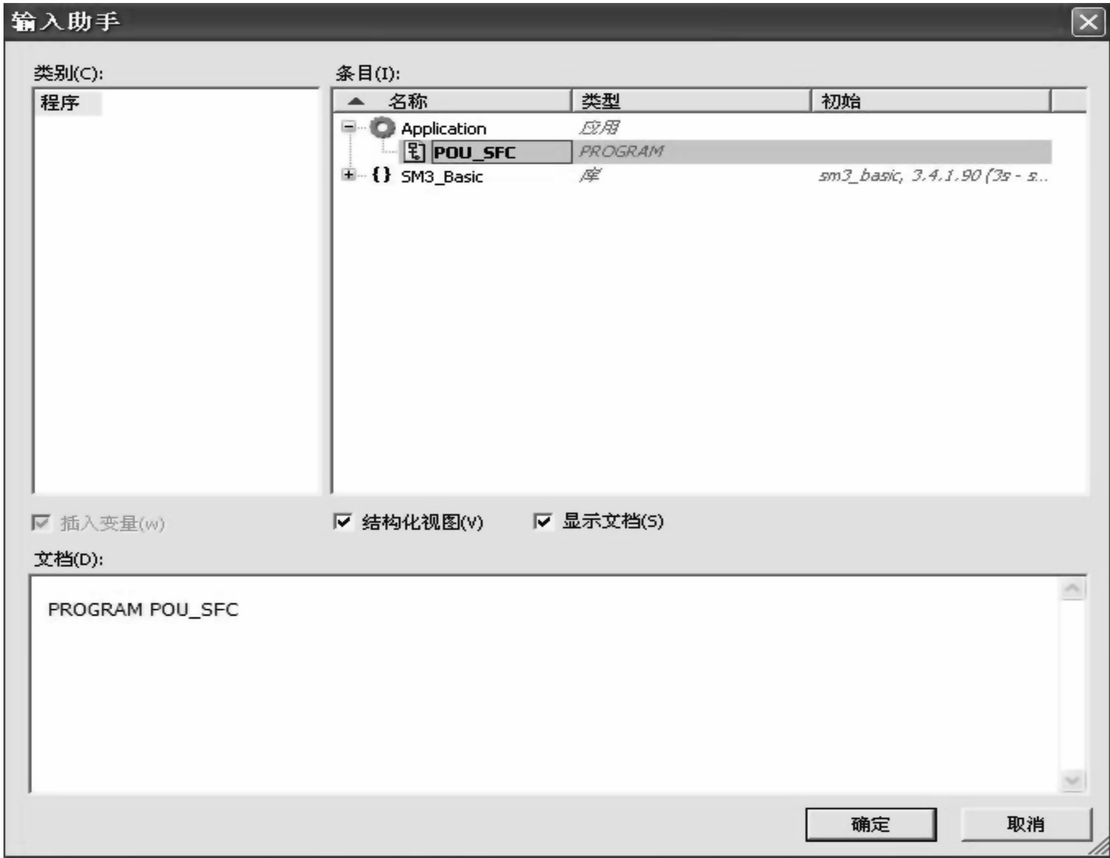


图 5-99　选择要执行的程序

这时，要执行的程序加入到 POU，如图 5-100 所示。按“F11”编译后，程序名由灰暗变亮，表示已加入扫描，如图 5-101 所示。

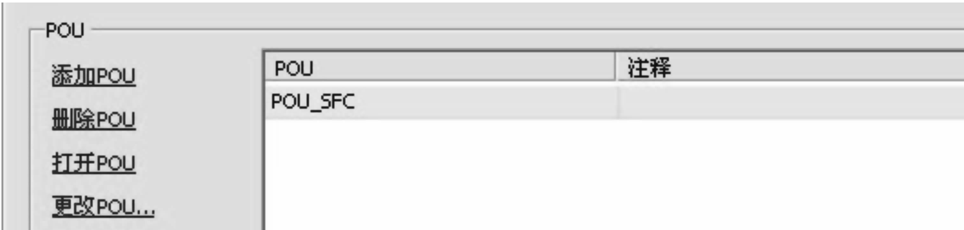


图 5-100　把要执行的程序加入到主任务

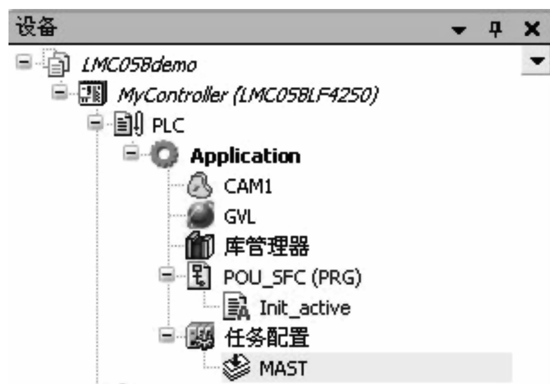


图 5-101　加入后的显示

启动仿真并登录运行，我们可以看到执行情况，如图 5-102 所示。

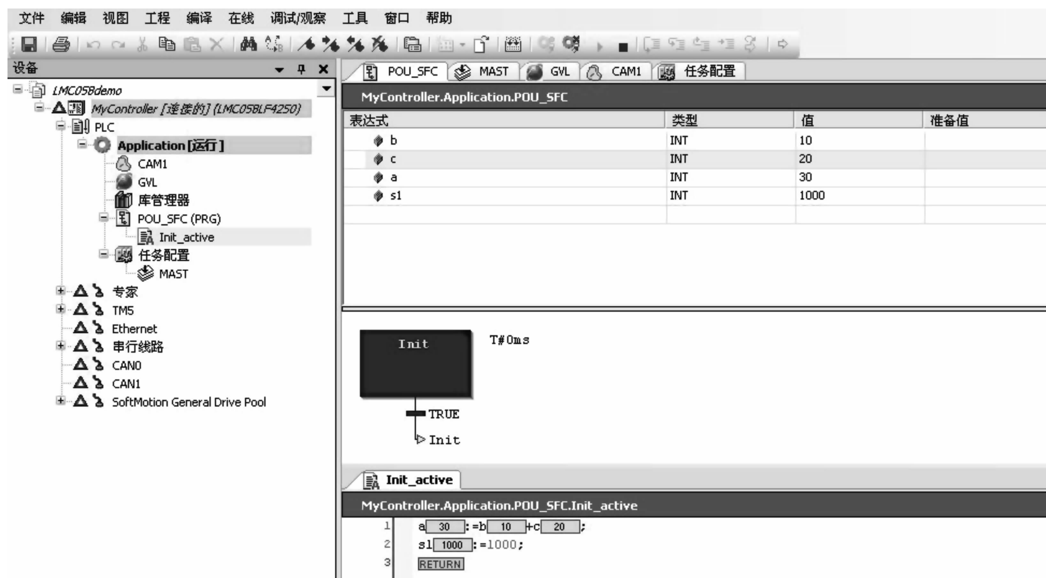


图 5-102　仿真执行程序

我们还可以在初始化方框内，点击鼠标右键，打开编辑功能菜单，添加入口动作和出口动作程序，如图 5-103 ~ 图 5-105 所示。

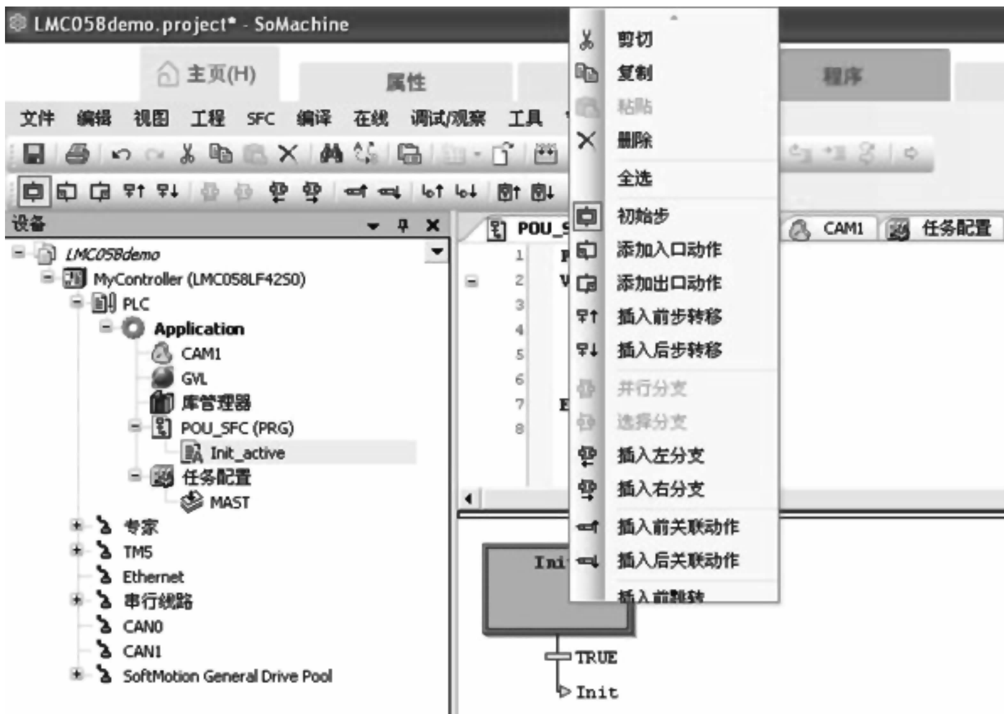


图 5-103　在功能框打开编辑菜单

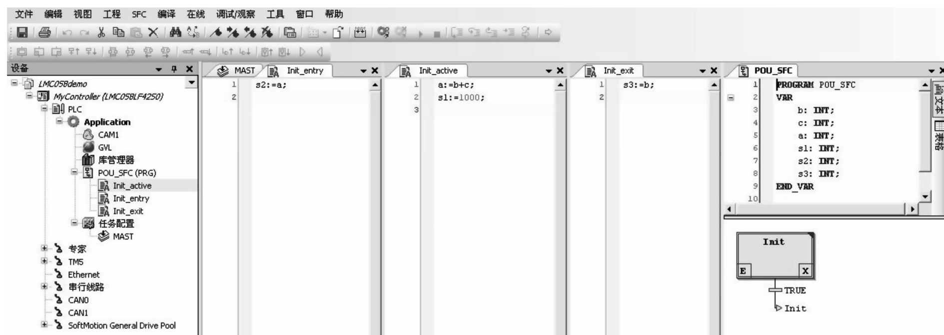


图 5-104　创建功能框入口和出口程序单元

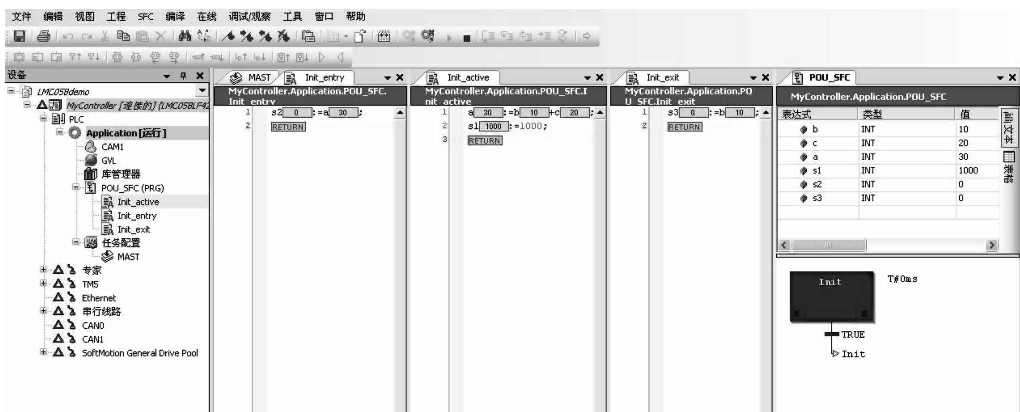


图 5-105　仿真执行一个功能框程序

在初始化方框内，点击鼠标右键，打开编辑功能菜单。选择插入后部转移，则加入一个程序块 Step0。把 Step0 上口的条件 trans0 写为程序开关 k1，下口条件写为开关 k2。用这两个开关，我们就可以控制程序执行流了。

闭合开关 k1 程序扫描离开初始化框，启动了 Init _ exit 程序，使 s3 被赋值，如图 5-106 所示。

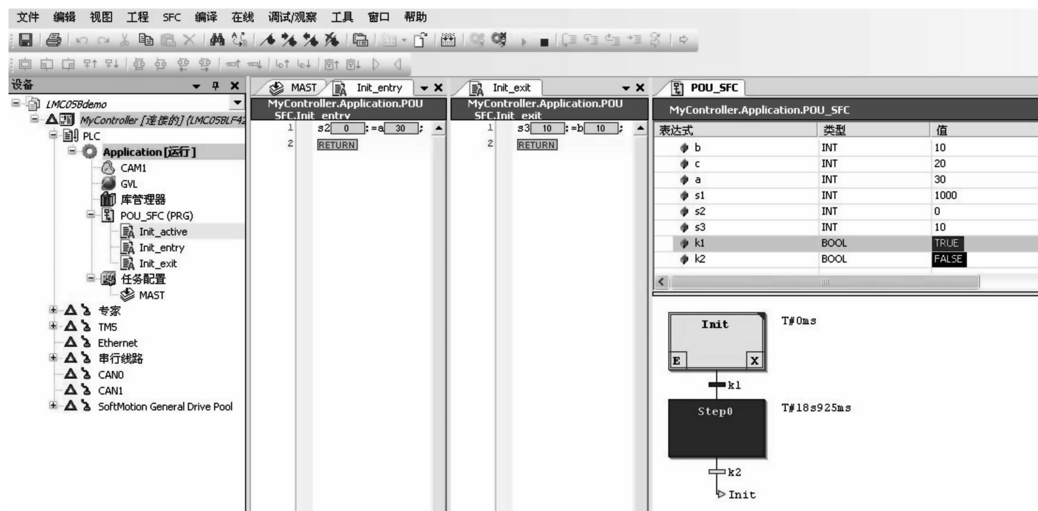


图 5-106　建立一个程序执行流程控制观察出口执行效果

打开 k1，闭合 k2，程序扫描离开 Step0，重新进入初始化框，启动了 Init _ entry 程序，使 s2 被赋值，如图 5-107 所示。

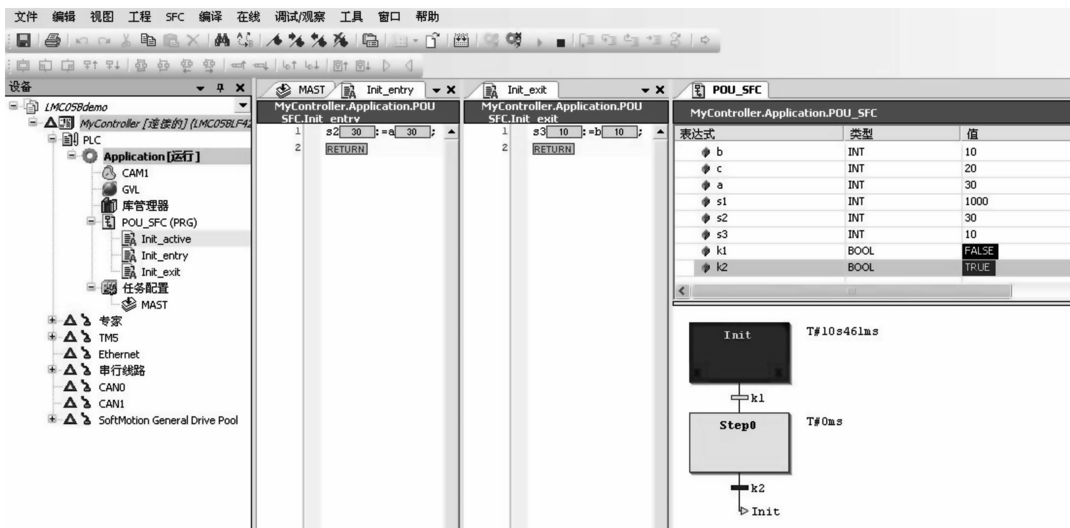


图 5-107　仿真流程控制观察入口效果

修改参数，再次闭合 k1，程序执行流又流到 Step0 可以看到各个程序的执行情况，如图 5-108 所示。

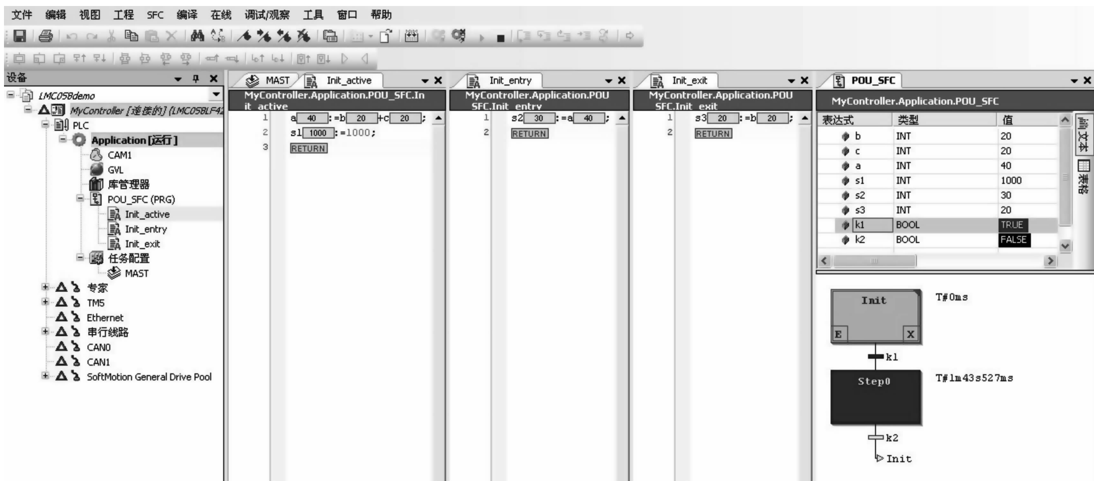


图 5-108　仿真程序流的控制

同理，在 Step0 框中，我们也可以建立入口执行程序、可执行程序和出口执行程序。也可以建立左分支程序和右分支程序。例如，在 Step0 我们建立一个右分支程序，即在框内点鼠标右键，打开编辑菜单，选择插入右分支后，即建立右分支程序框 Step1，如图 5-109 所示。

注意：这两个程序框是双线连接，因此对程序的扫描是并行的。如果想让程序的执行非此即彼，则要用鼠标点在双线处，激活图标行然后点击“选择分支”图标。但要注意的是：一旦选择了“选择分支”，则分支的各个程序框的入口和出口都必须要有条件开关来守门，如图 5-110 所示。

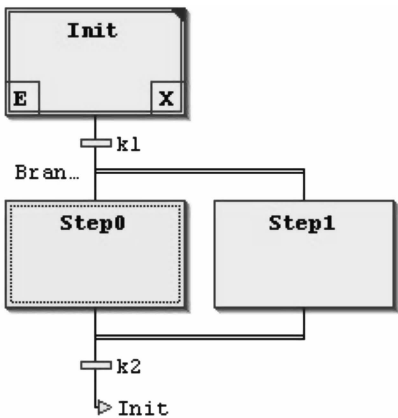


图 5-109　向右扩展一个程序框

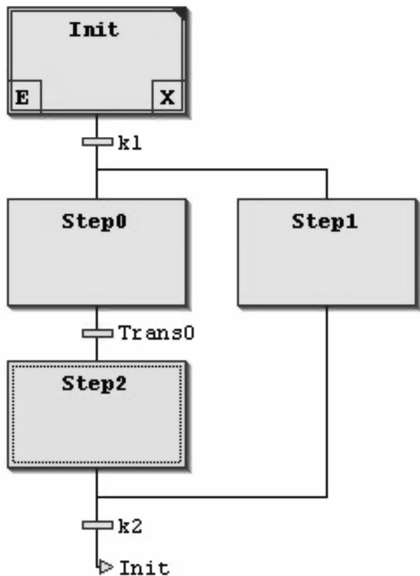


图 5-110　有条件开关守门

向下插入一个程序框，得到一个条件转移 Trans0 和程序框 Step2，如图 5-111 所示。

向上插入一个程序框，得到一个条件转移 Trans1 和程序框 Step3。

同理在 Step1 上编辑，加入 Step4、Step5 功能框，得到 Trans2、Trans3 条件判断，如图 5-112 所示。

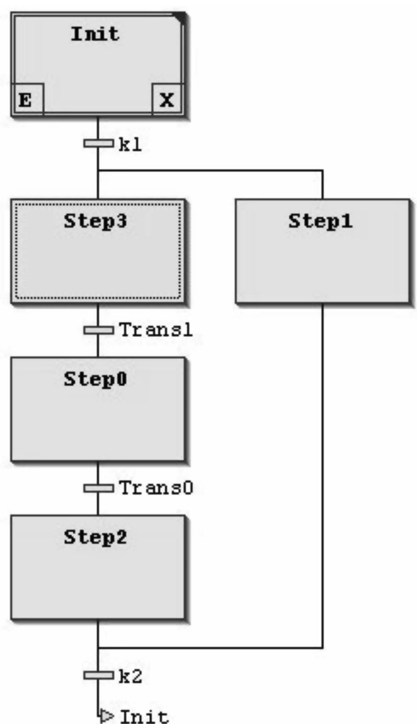


图 5-111　向下插入一个程序框

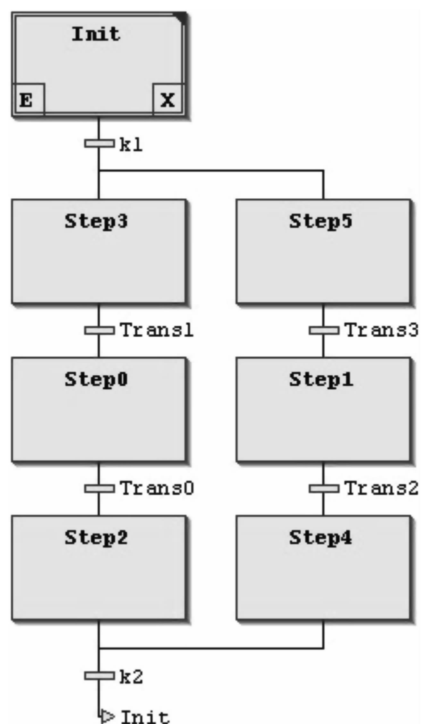


图 5-112　加入 Step4、Step5 功能框，得到 Trans2、Trans3 条件判断

当我们把 Step2、Step3、Step4、Step5、k1、k2 删除，得到如下符合编程语法的分支执行结构，如图 5-113 所示。

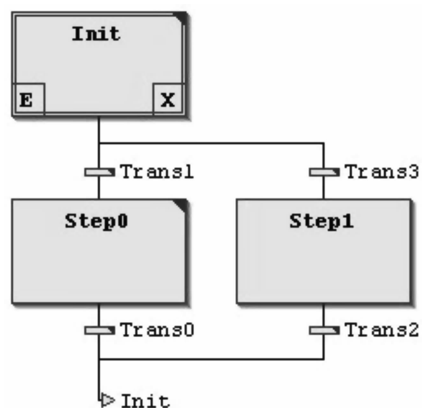


图 5-113　符合编程语法的分支执行结构

注意，如果不删除 k1、k2，在编译时会出现错误。

双击 Trans1，我们可以创建一个条件转移程序，程序的编程方式可以在菜单中选择。例如，选择结构文本 ST，如图 5-114 所示。

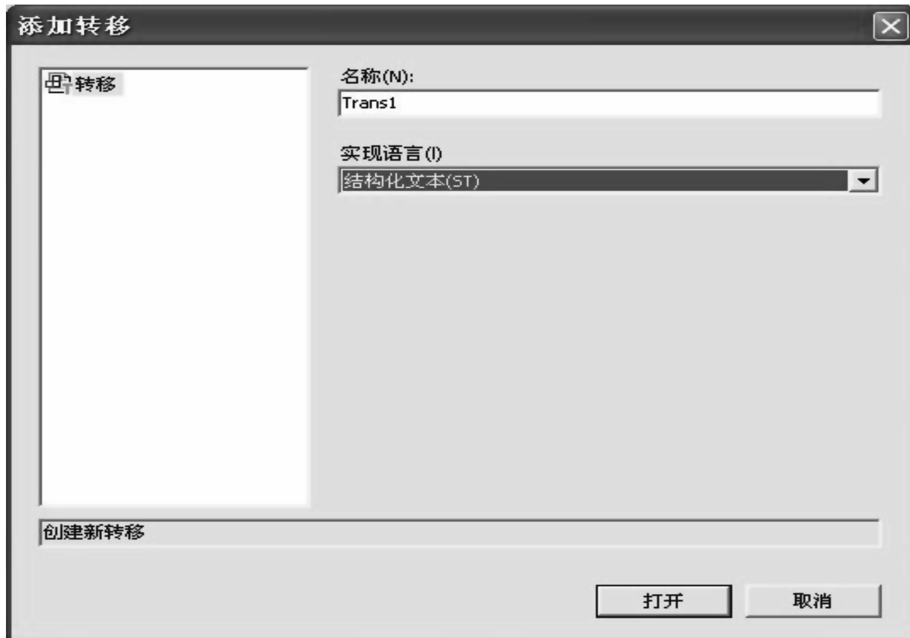


图 5-114　选择结构文本 ST

打开后，在编程区编辑一个比较语句，如果语句为真，则进入下边程序框，如图 5-115 所示。

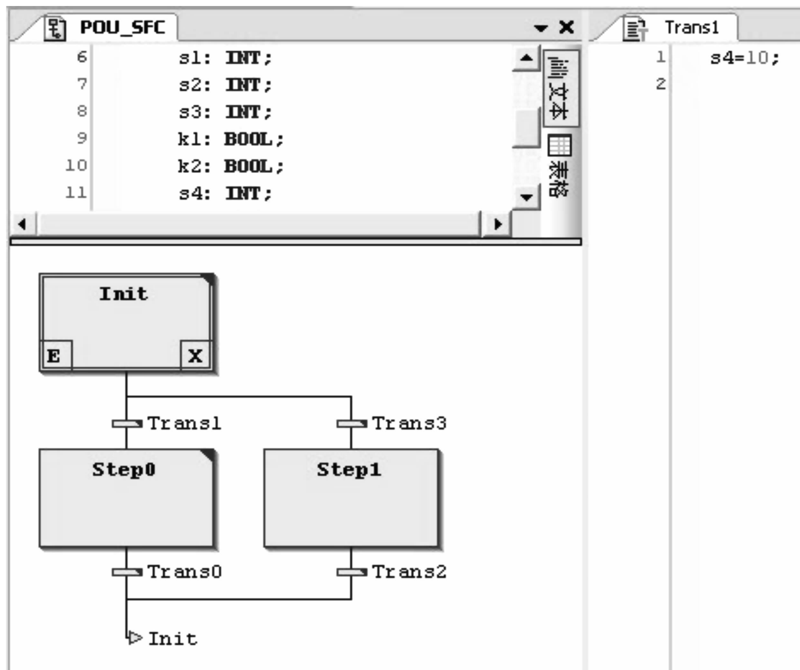


图 5-115　建立一个条件转移程序 Trans1

同理，我们可以建立条件转移程序 Trans0、Trans2、Trans3，如图 5-116 所示。

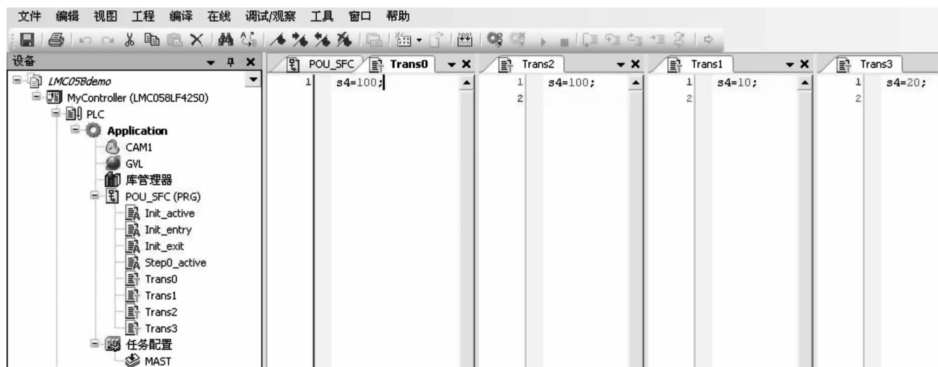


图 5-116　建立多个条件转移程序

我们开始仿真程序，初始程序流状态如图 5-117 所示。

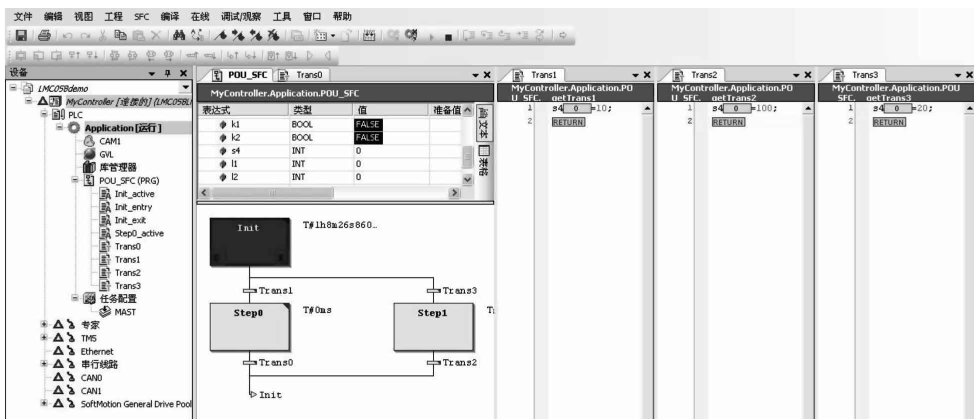


图 5-117　初始程序流状态

给 S4 赋值，使得条件满足，进入 Step0，如图 5-118 所示。

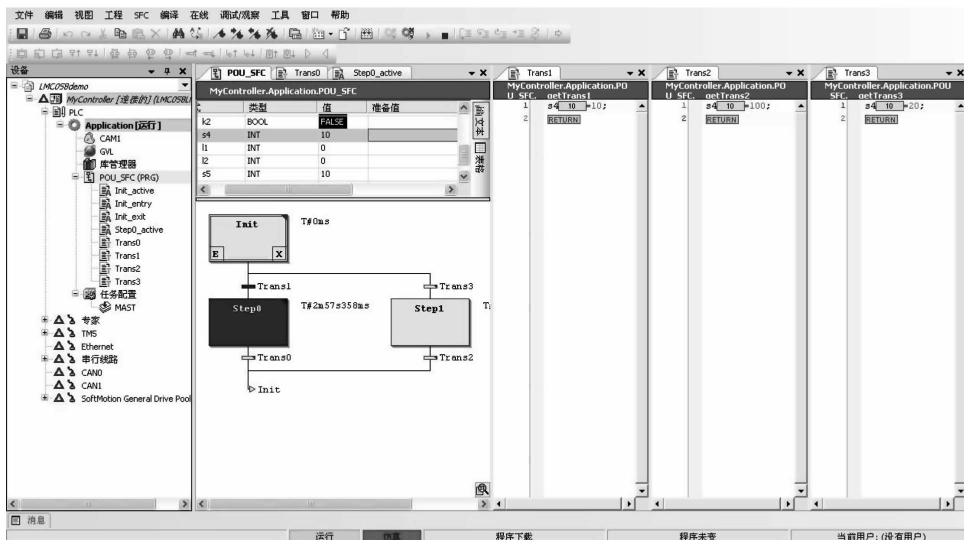


图 5-118　满足条件后进入下一步程序块

给 S4 赋值 100，使之满足出口条件，导引程序流出，如图 5-119 所示。

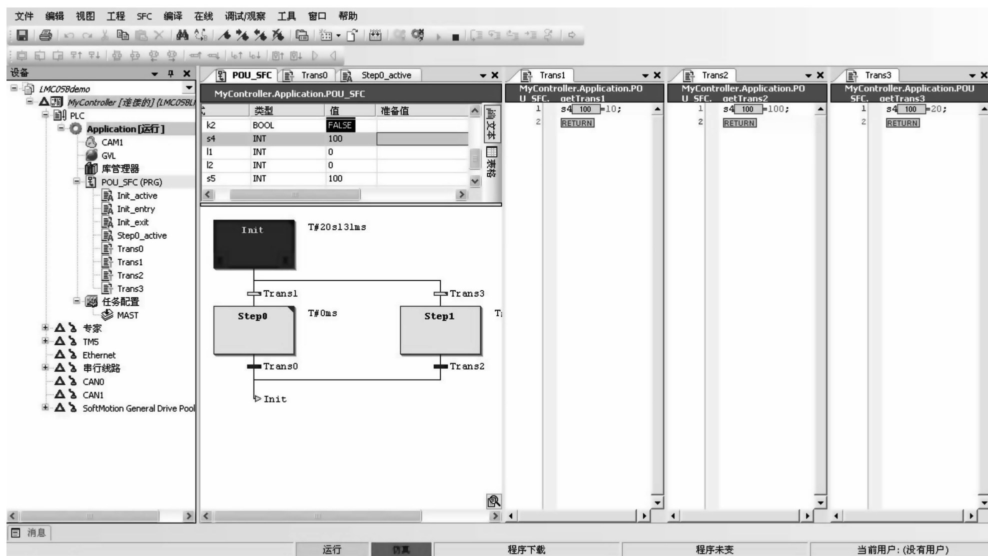


图 5-119　满足出口程序条件

给 S4 赋值 20，使之满足入口程序 Trans3 的条件，导引程序流进 Step1，如图 5-120 所示。

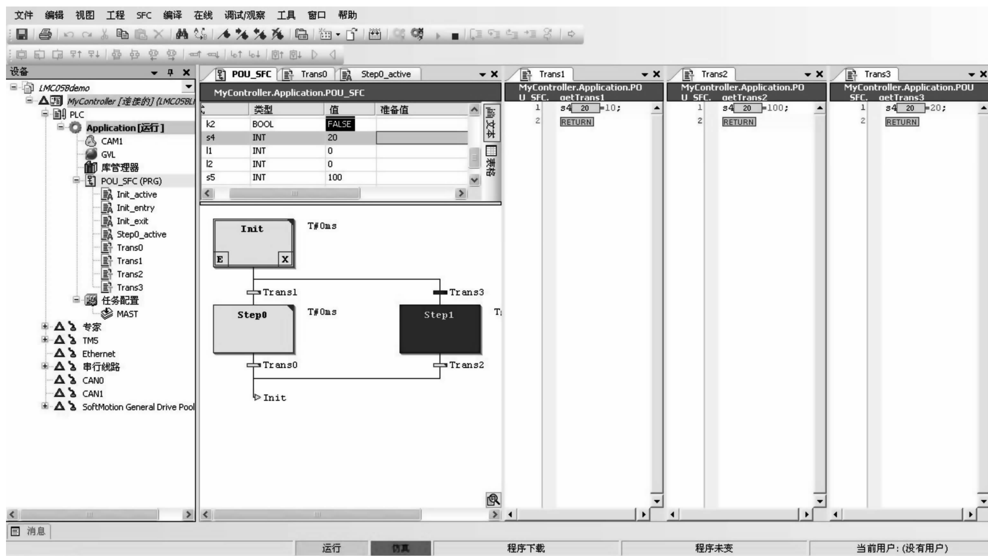


图 5-120　满足 Trans3 条件判断

综上所述，从这个程序案例来看，我们可以通过条件判断或程序运算的生成结果来控制程序的执行流程或决定哪些程序块投入运行，就像警察指挥交通一样，使你的程序执行效率大大提高。

5.6　功能块的建立

当一个功能在程序中重复使用时，我们有必要建立一个功能块，这样有利于编程的简

捷。例如已知的功能块：计时器，计数器功能块。

功能块（FB）中，可以有多个输入和输出管脚；变量结构与这个功能块相关联；功能块存储执行中得到的内部变量结果。

例如，我们建立一个功能块 My _ FB，如图 5-121 所示。



图 5-121　功能块的建立步骤

如图 5-121 所示，我们按照步骤，添加程序组织单元 POU 并在此画面上选择功能块形式和命名功能块，然后选择编辑此功能块的语言。打开后，如图 5-122 所示，在变量区，定义功能块的输入、输出管脚并且声明管脚类型。在编程区编写内部关联。

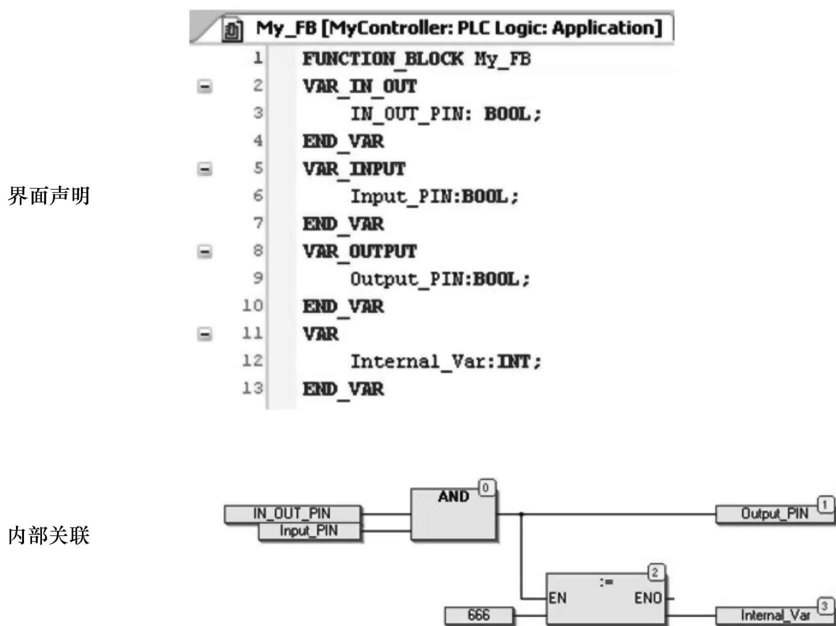


图 5-122　功能块的编写

在建立好功能块后，在设备区的应用栏目下会出现功能块的程序名，如图 5-123 所示。

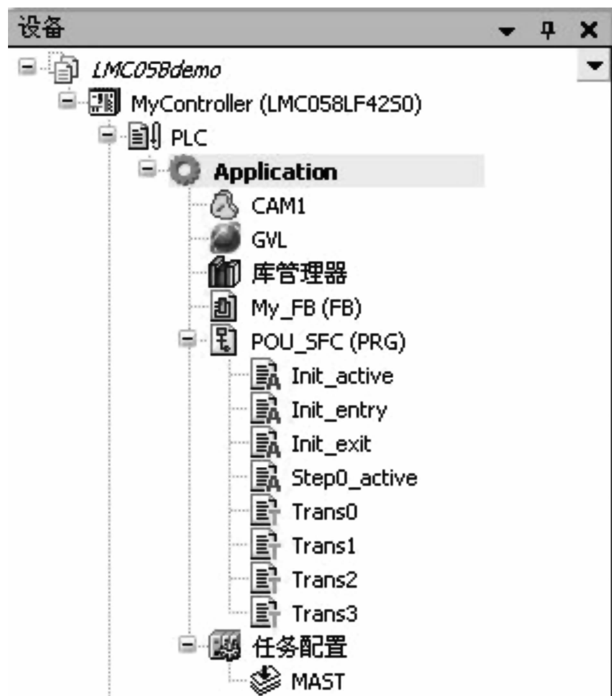
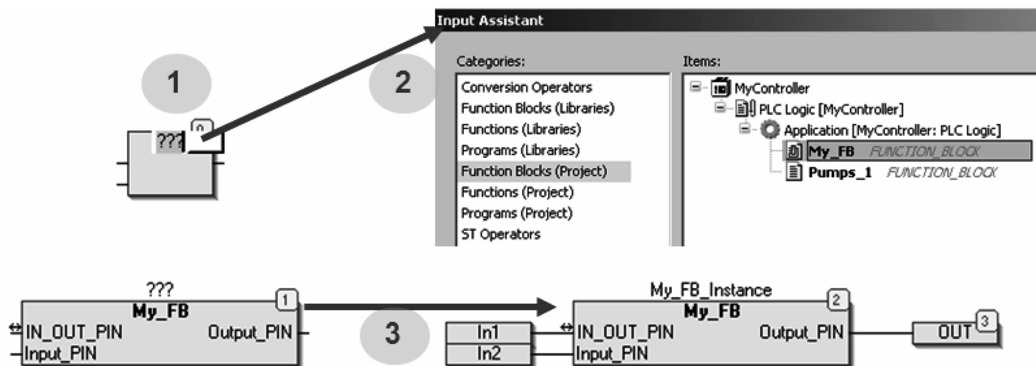


图 5-123　显示在应用层的功能块

在编写程序要调用此功能块时，先添加一个运算块，然后点击白色小框打开输入帮助，在应用程序层选择编好的功能块并命名即可应用。功能块的调用步骤如图 5-124 所示。



● 添加运算块并打开输入帮助 (1)

● 选择已定义功能块 (2)

● 添加功能块名字和变量 (3)

图 5-124　功能块的调用步骤

第 6 章　数 据 类 型

6.1　支持的数据类型，命名规则

6.1.1　支持的数据类型概览

1. IEC 标准数据类型

例如：布尔（BOOL），整数（INT），实数（REAL），字符串（STRING）等，见表 6-1。

表 6-1　标准数据类型

关 键 字	前缀 ***	位数（位）	关 键 字	前缀 ***	位数（位）
BOOL	×	8	DWORD	dw	32
SINT	si	8	LWORD	lw	64
INT	i	16	REAL	r	32
DINT	di	32	TIME	tim	32
LINT	li	64	LTIME*	ltim	64
USINT	usi	8	TOD	tod	32
UINT	ui	16	DATE	date	32
UDINT	udi	32	DT	Dt	64
ULINT	uli	64	STRING(XX)**	s	
BYTE	by	8	WSTRING(XX)**	ws	
WORD	w	16			

2. 数据单元类型（DUT）

可以建立的数据单元类型有：数组变量（Array），结构变量（Structure）和枚举（Enumeration）变量。

3. 引用（References）IEC 标准数据类型的扩展

引用（References）是对一个项目别名的定义。这个别名可以通过标识符读写。

4. 变量类型

局域变量（Local Variables）和全局变量（Global Variables）。

6.1.2　变量命名规则

1. 标准变量的命名

变量名由两部分组成，即 <前缀> <基本名>。前缀用来指示数据类型，剩下的就称为基本名。基本名可以很短，是变量功能的意义解释。

例如：变量名bySubindex：表示这个变量是一个子索引命令，类型为字节。

sFileName：表示这个变量是一个文件名，类型为字符串。

iCounter：表示是一个计数变量，类型为整数 INT。

变量命名的限制性规定如下：

- 变量名只能有单下画线；
- 不能有空格；
- 不能有 IEC 关键字和操作符；
- 不能有特别字符如 ＋， －. ＊，/…
- 不能与功能块同名。

2. 引用（Reference）和指针（Pointer）

引用是用来给标识符定义数据类型的，它的语法结构为：

标识符：REFERENCE TO 数据类型；

这样就在变量声明区完成了对标识符的数据类型声明。也就意味着这个标识符可以是所有具有同样数据类型的变量的别名。当在程序编辑区执行程序时，可以使用以下语法结构来把其他同类数据类型变量与标识符关联起来。

标识符 REF = 同类型变量。

引用的使用如图 6-1 所示。



图 6-1　引用的使用

有效引用的检查：

操作符“_ ISVALIDREF”可用来检查引用是否指向一个不等于 0 的有效值。

语法：

布尔变量：＝__ ISVALIDREF（标识符）；

如果引用指向一个有效值，则＜布尔变量＞为真（TRUE），否则为假（FALSE）。

有效引用的检查如图 6-2 所示。

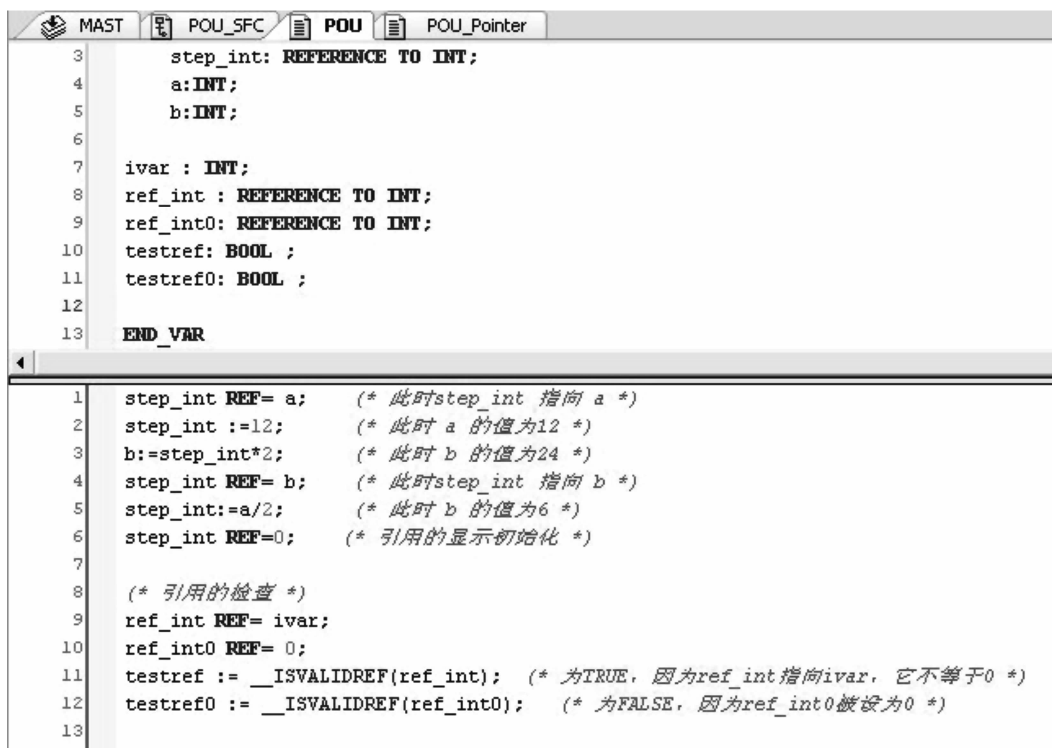


图 6-2　有效引用的检查

引用检查的仿真如图 6-3 所示。

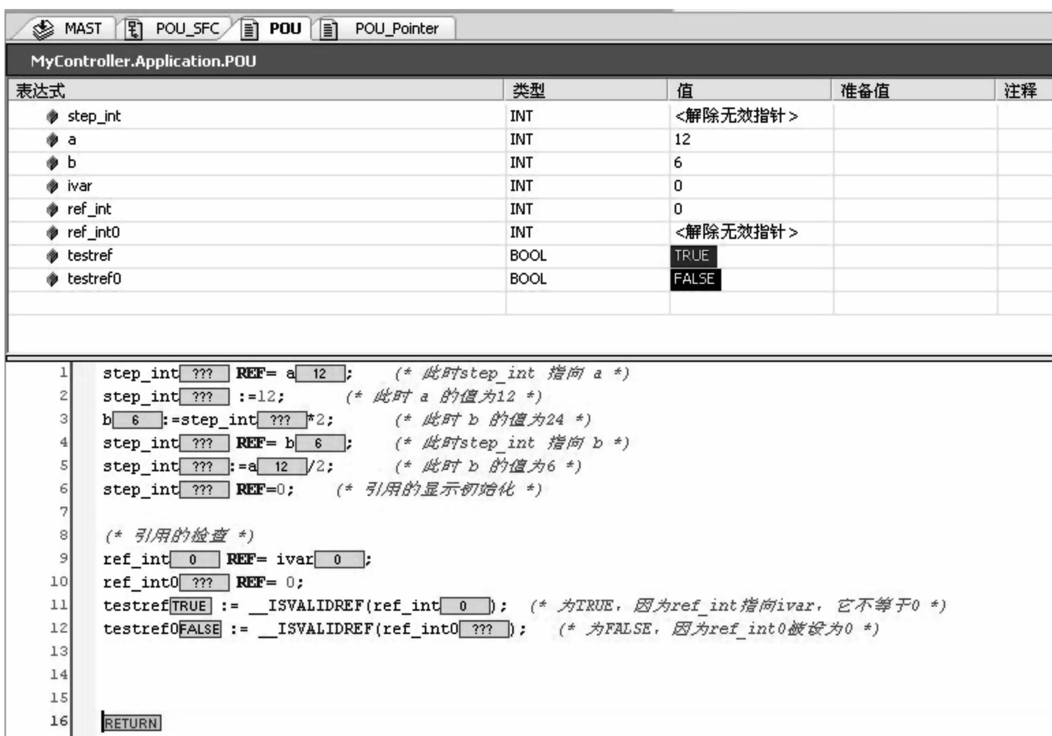


图 6-3　引用检查的仿真

指针（Pointer）

作为 IEC 61131-3 标准的扩展，您可以使用指针。

指针用来在应用程序运行时存储变量、程序、功能块、方法和函数的地址。它可以指向上述的任何一个对象以及任意数据类型，包括用户定义数据类型。请注意，您可以使用隐含的指针校验功能。

声明指针的语法如下：

<标识符>：POINTER TO <数据类型|功能块|程序|方法|函数>;

取指针地址内容即意味着读取指针当前所指地址中存储的数据。通过在指针标识符后添加内容操作符“^”，可以取得指针所指地址的内容：请看下面“pt^”的使用示例。

通过地址操作符 ADR 可以将变量的地址赋给指针。

指针的应用如图 6-4 所示。

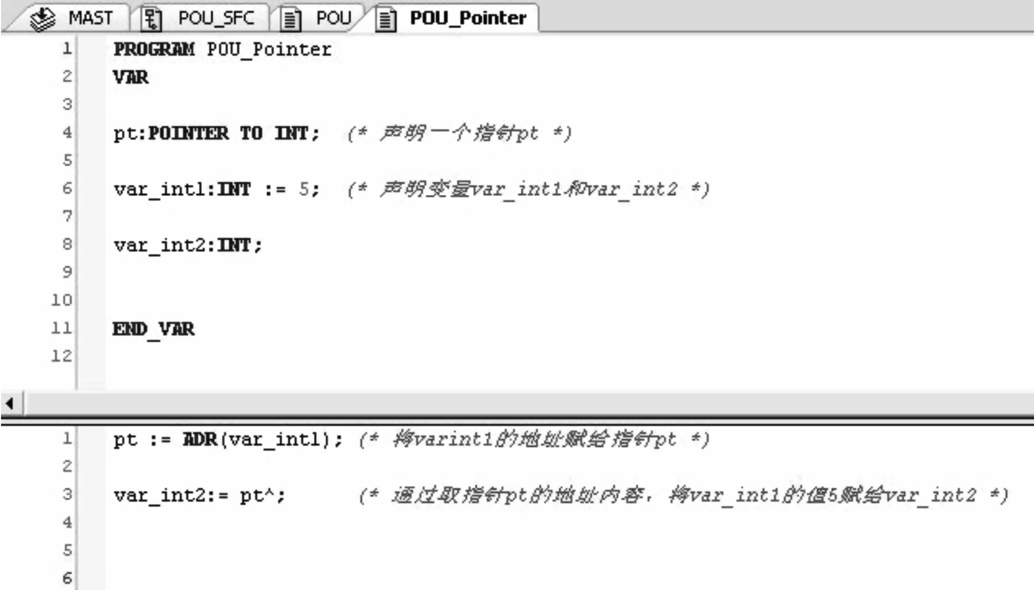


图 6-4　指针的应用

指针的仿真运行如图 6-5 所示。



图 6-5　指针的仿真运行

函数指针

与 CoDeSys V2.3 不同的是，本版本用函数指针取代了 INDEXOF 操作符。这类指针可以指向外部库，但不可以在编程系统的应用程序内部调用函数指针！注册回调函数（系统库函数）的运行时函数需要函数指针，之后根据注册所需的回调函数的不同，各自的函数将由运行时系统隐式调用（例如 STOP）。若需要使能这样的系统调用（运行时系统），必须为函数对象设置各自的属性（类别“编译”）。

ADR 运算符可以用于函数名、程序名、功能块名和方法名。由于函数在在线修改之后有可能被移动，因此运算结果不是函数的地址，而是指向函数的指针的地址。只要函数存在，这一地址就将一直有效。

指针的索引访问

作为 IEC 61131-3 标准的扩展，允许对 POINTER、STRING 和 WSTRING 等类型的变量进行索引访问“［］”。

■ pint［i］返回被指向的数据类型。

■指针的索引访问的算法：如果对指针变量进行索引访问的操作，则偏移量将按照下式计算：pint[i] =(pint + i * 被指向的数据类型的长度)^。索引访问也隐含了获取指针地址内容的操作。所得结果的数据类型是指针所指的数据类型。请注意：pint[7]！ =(pint +7)^!

■如果对 STRING 类型的变量进行索引访问的操作，则会得到索引表达式偏移量处的字符。所得结果是 BYTE 类型。str[i] 将会以 SINT 的形式（ASCII 编码）返回字符串中的第 i 个字符。

■如果对 WSTRING 类型变量进行索引访问的操作，则会得到索引表达式偏移量处的字符。所得结果是 WORD 类型。wstr[i] 将会以 INT 形式（Unicode 编码）返回字符串中的第 i 个字符。

注意：请注意引用与指针不同，它们将直接修改所指的数据。

指针校验函数（CheckPointer function）

为了在程序运行时检查指针的指向，您可以在每次访问指针的地址之前使用隐含的“指针校验”功能。您可以通过添加对象对话框向应用程序中添加“用于隐含检查的 POU”对象。点选指针校验的复选框，选择一种实现语言，确认无误后点击打开。校验功能将会在编辑器中以您选择的语言打开。声明部分是预先设置的，与前述选项的选择无关，并且只有添加了其他局部变量之后才可以更改！与其他校验函数不同的是，没有提供指针校验函数的默认实现，这一部分需要由用户编写。

指针校验函数需要检查指针指向的地址是否在有效的存储范围之内，另外还需要检查引用的连续内存空间与指针所指的变量的数据类型是否匹配。若满足上述两个条件，指针校验应当返回这个输入指针。出现错误时则交由用户进行适当的处理。

模板

声明部分：

```
//隐含生成的代码：请勿修改
FUNCTION CheckPointer：POINTER TO BYTE
VAR_INPUT
    ptToTest：POINTER TO BYTE;
```

```
    iSize: DINT;
    iGran: DINT;
    bWrite: BOOL;
END _ VAR
```

实现部分:

```
//没有标准的实现。请在此处添加您编写的代码
CheckPointer: =ptToTest;
```

调用本函数时需要以下输入参数:

- ptToTest:指针的目标地址。
- iSize:引用的变量的长度;iSize 的数据类型必须与整型兼容且能涵盖存储在指针地址中的可能的最大数据长度。
- iGran:存取的单位长度,如用于被引用变量的最大的未结构化的数据类型;iGran 的数据类型必须与整型兼容。
- bWrite:读取方式(TRUE 为写方式,FALSE 为读方式);bWrite 必须是布尔类型返回值:取指针地址内容时使用的地址,因此最好是第一个输入参数的地址(ptToTest)。

6.2 局域变量(Local)和全局变量(Global)

变量的声明

每一个程序组织单元 POU 都有一个变量声明区。

变量声明的语法结构为

变量:类型:=初始化数值。

变量声明的语法结构如图 6-6 所示。

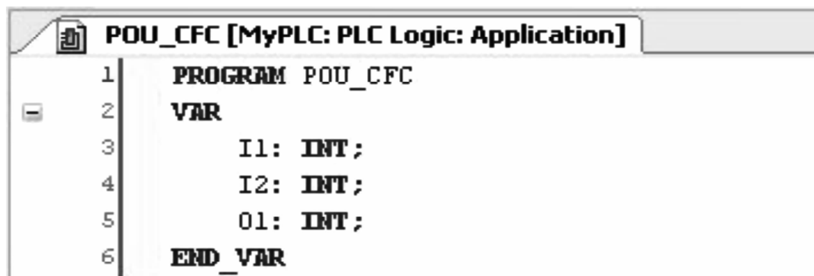


图 6-6 变量声明的语法结构

多个变量可以如下声明:iVar1, iVar2, iVar3: INT;

变量范围由以下开始:

- VAR;
- VAR _ INPUT;
- VAR _ OUTPUT;
- VAR _…

由 END _ VAR 结束。变量声明如图 6-7 所示。

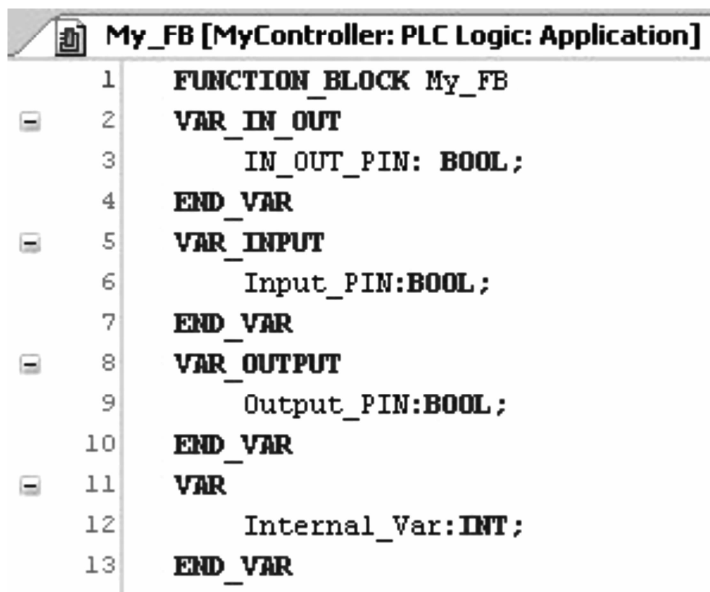


图 6-7　变量声明

变量声明-局域变量

局域变量在程序组织单元 POU 的变量名声明区声明。

它们可以被其他 POU 调用（取决于变量范围），如图 6-8 所示。

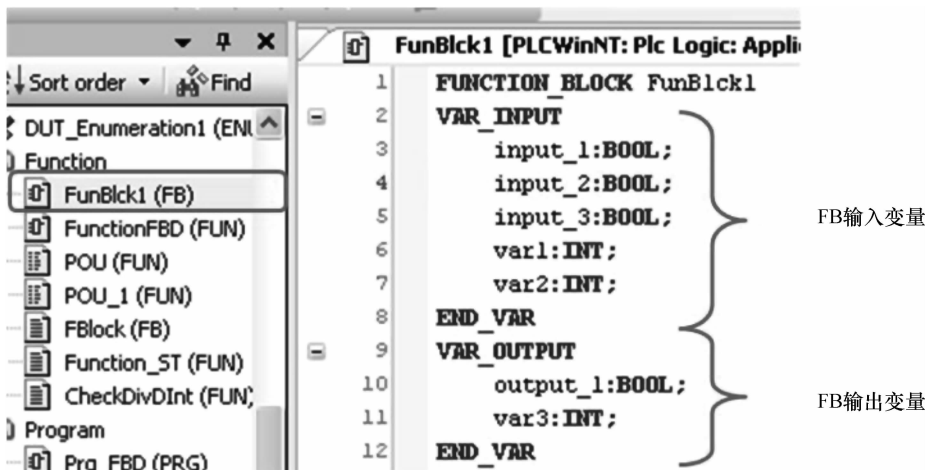


图 6-8　变量可由其他 POU 调用

变量声明-全局变量

全局变量在全局变量列表（GVL）中声明，如图 6-9 所示。

它们可以在所有应用中被访问。

每当键入完一个新的变量都会自动弹出一个声明框，如图 6-10 所示。

- CONSTANT：初始化常数。
- RETAIN：保持型变量可以在 POU 或全局变量列表中使用关键字“RETAIN”来声明此类变量。

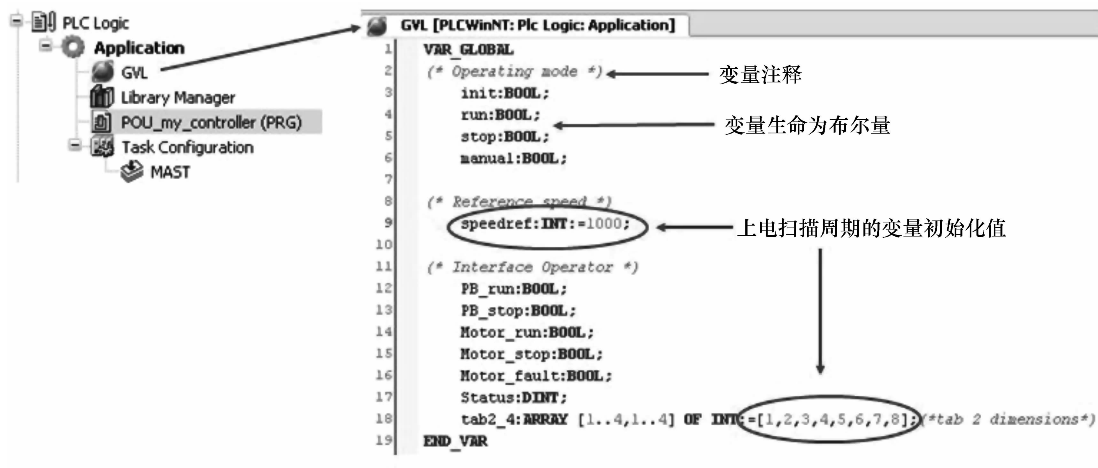


图 6-9　全局变量声明

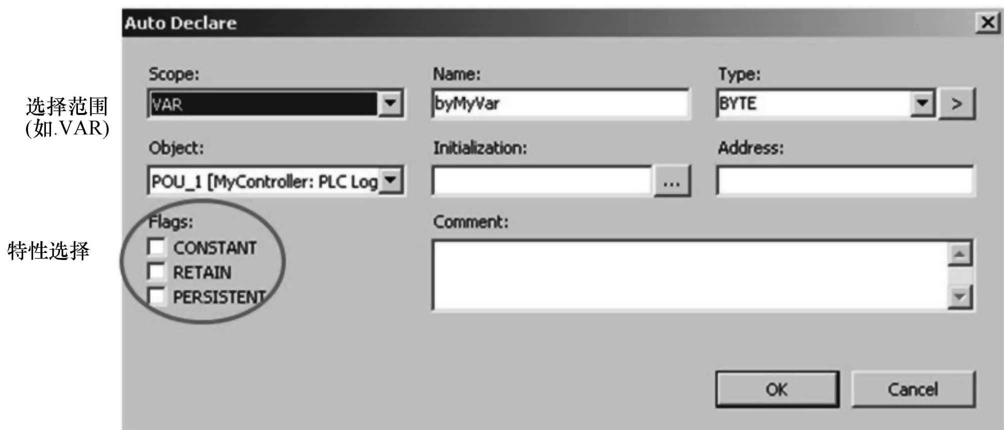


图 6-10　自动变量声明

例如：

VAR RETAIN

　iRem1：INT;（＊1. Retain 变量＊）

END _ VAR

Retain 属性：以关键字 RETAIN 声明 Retain 类型变量。Retain 型变量在控制器正常关闭、打开（或收到在线命令“热复位”），甚至意外关闭之后这类变量仍然能保持原来的值。随着程序重新开始运行，存储的值能继续发挥作用。生产线上的计件器便是一个典型的例子：电源被切断之后，它仍然可以在再次启动时继续计数。而其他所有变量此时都将被重新初始化，变为指定初始值或标准初始化的值。

与 Persistent 变量不同的是，Retain 变量将会在新的程序下装之后被重新初始化。

应用举例：生产机器中的计件器，在掉电之后可以继续计数。

但 Retain 变量在“初始化复位”、“冷复位” 和程序下装之后将会重新初始化；而相对地，persistent 变量只在“初始化复位” 之后重新初始化。

内存位置：依赖于设备。

1）Retain 型变量存储于标准数据存储区，并且被循环复制到一个 Retain 内存中。

2）Retain 型变量仅仅被存储在一个单独的内存区中。

- PERSISTENT：变量的重新初始化按需要进行。

由关键字“PERSISTENT”识别 Persistent 变量（VAR _ GLOBAL PERSISTENT）。它们仅在（应用）重设起点时被重新初始化。与 Retain 变量相比，它们在下装后仍保持原有的值。一个 Persistent 变量的应用例子是操作时间计数器，即使在断电和下装以后它仍然应该继续计数。见下面的表 6-2。

Persistent 变量的处理情况与 CoDeSys V2. 3 中有所不同，如下所述。

Persistent 变量只能在特定的“Persistent 变量”对象类型全局变量列表中声明，这一类型是指定一个应用的。每一个应用之中可能只有一个这种列表。

注意：从 V3. 3. 0. 1 版开始用“VAR _ GLOBAL PERSISTENT”声明，这与“VAR _ GLOBAL PERSISTENT RETAIN”或“VAR _ GLOBAL RETAIN PERSISTENT”声明方法等效。

与 Retain 变量一样，Persistent 变量也存储在一个独立的内存区中。

例如：

```
VAR GLOBAL PERSISTENT RETAIN
   iVarPers1：DINT;（ *1. App1 中的 Persistent + Retain 变量 * ）
   bVarPers：BOOL;（ * 2. App1 中的 Persistent + Retain 变量 * ）
END _ VAR
```

注意：目前只有全局 Persistent 变量可以这样。

目标系统必须提供一个独立的存储空间来保存每一个应用的 Persistent 变量列表。

每一次重新装载应用时，PLC 中的 Persistent 变量列表将与工程中的列表核对。此时通过应用的名称识别 PLC i. a. 中的列表。一旦两者不一致，将会提示用户重新初始化应用中所有的 Persistent 变量。这种错误可能是由于对已经存在于列表中的声明进行重命名、移除或其他操作造成的。

用户只能在列表的末尾添加新的声明。在下装过程中，这些声明将会被识别为“新”声明，而不会造成整个列表的重新初始化

表 6-2　保持型变量的行为概述表

×—保持原值　— = 被初始化

after Online commandl 收到在线命令之后	VAR	VAR RETAIN	VAR PERSISTENT	VAR PERSISTENT VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN
Reset Warm < application > 热复位	—	×	×	×
Reset Kalt < application >	—	—	×	×
Reset Ursprung < application >	—	—	—	—
Download < application > 下装	—	—	×	×
Online Change < application > 在线修改	×	×	×	×
Reboot PLC 重新启动 PLC	—	×	—	×

6.3 数据单元类型（Data Unit Type）

6.3.1 数组变量（Array）

- 最大为3维。
- 声明语法结构：
  变量：ARRAY[0…X]OF 类型。
- 自动声明启动数组向导，如图6-11所示。

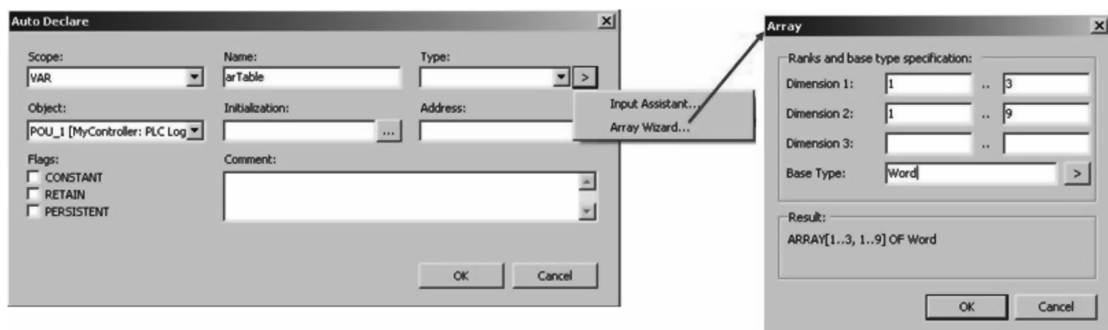


图6-11　数组的自动声明向导

图6-12所示：具有3列，9行的2维数组。

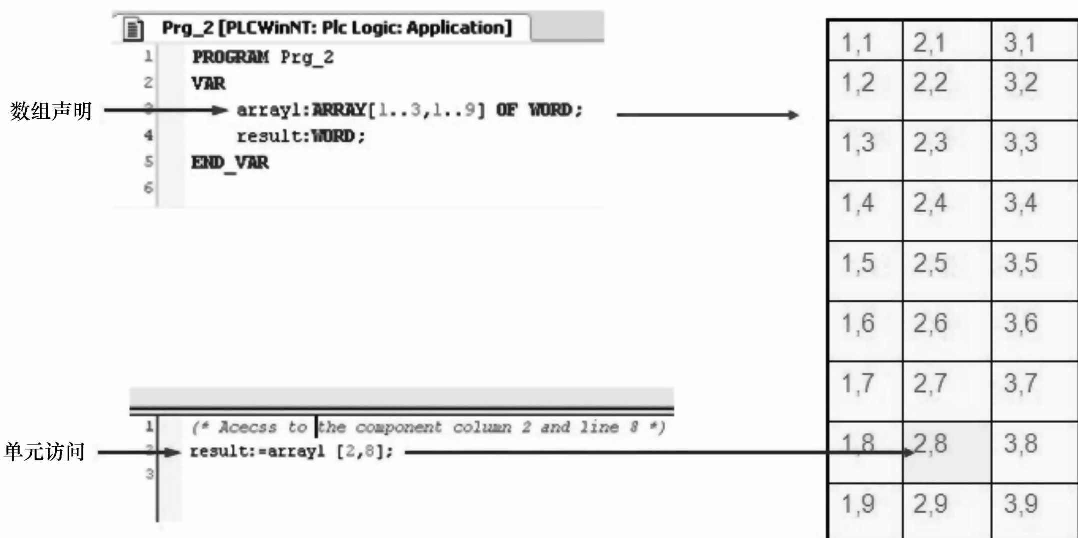


图6-12　2维数组的元素分布

采用数组偏置的间接寻址如图6-13所示。

Array_Offset [MyController: PLC Logic: Application]

MyController.Application.Array_Offset

Expression	Comment	Type	Value
arMyArray		ARRAY [0..9] OF WORD	
arMyArray[0]		WORD	0
arMyArray[1]		WORD	0
arMyArray[2]		WORD	0
arMyArray[3]		WORD	0
arMyArray[4]		WORD	44
arMyArray[5]		WORD	0
arMyArray[6]		WORD	0
arMyArray[7]		WORD	0
arMyArray[8]		WORD	0
arMyArray[9]		WORD	0
iIndex		INT	4
wElement		WORD	44

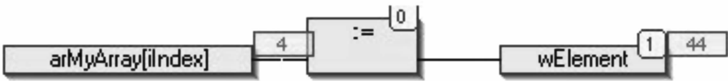


图 6-13　数组的间接寻址

6.3.2　结构变量

结构变量（Structure）就像我们把人群按家族、组织、爱好划分为一个个群体一样。当我们感兴趣一个群体或一个家族时，我们就可以单独用这个群体或家族的一个名字来找到这个名下的所有成员。例如：如图 6-14 所示，在 MyStruct 名下，有各种不同类型的变量，但找到它们都要通过 MyStruct。

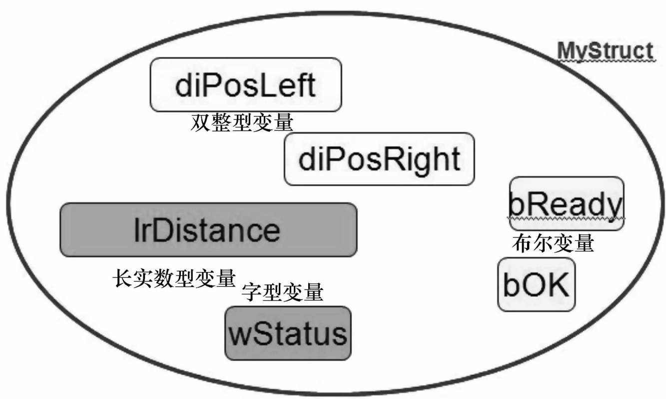


图 6-14　结构变量

在命名一个结构变量时，我们要按如图 6-15 所示步骤进行。即先打开栏目菜单的工程（Project）选择添加对象；在添加对象条目中，选择 DUT，并写上结构体变量名；打开结构体后，在声明区填写结构体名下的变量及类型。

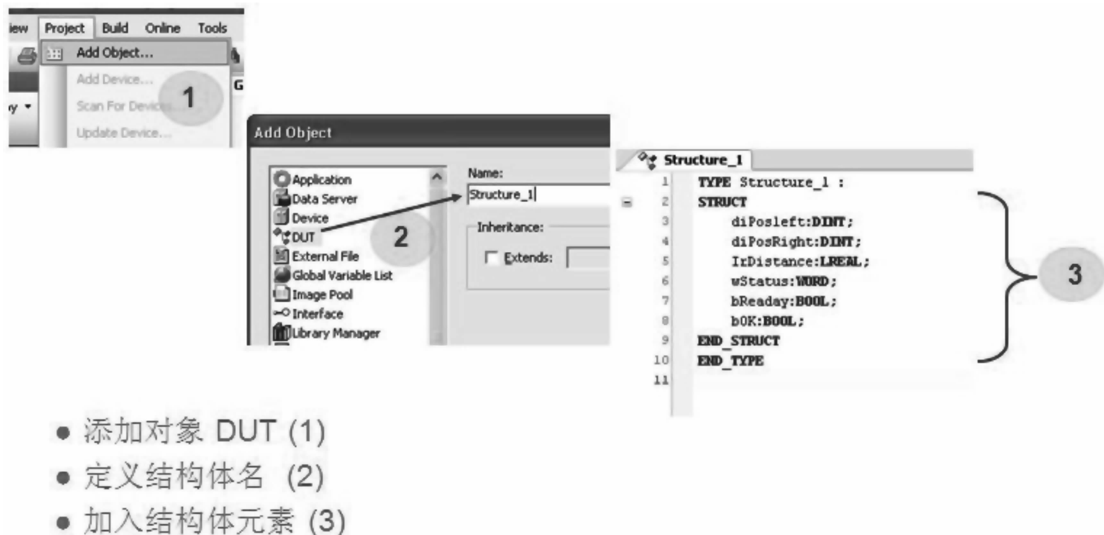


图 6-15　命名一个结构变量的步骤

编写完成后，就形成了在这个结构体下的一组或一个群体的变量。这个群体的代言就是结构体名。当在某一个程序组织单元 POU 中，要用到这组变量时，我们就可以在 POU 的变量声明区中把某一变量声明为这个结构体，然后这个变量就可以访问结构体变量下的元素了。

例 1：在编写一个程序组织单元时，首先声明变量为编辑好的结构体变量（1）即声明变量 tooldata 为结构变量 structure _ 1，如图 6-16 所示。然后在程序编辑区应用结构体变量时，先写出结构变量 tooldata，然后在其后加“.”，就引导出结构体下的所有元素（2）。

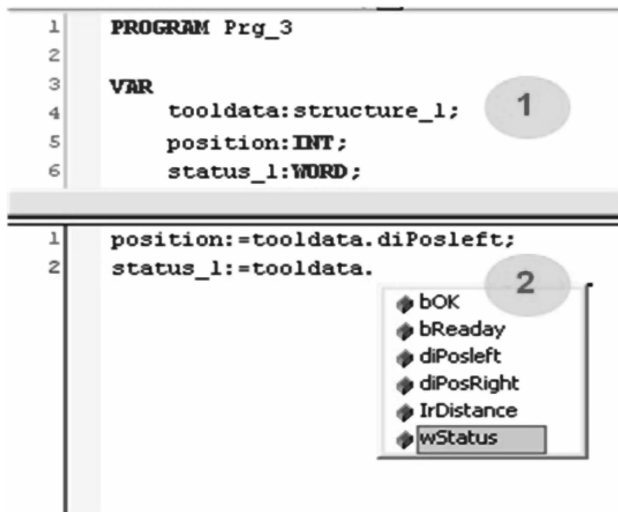


图 6-16　一个结构体变量下的所有元素

例 2：在实际电动机轴的运动控制中，这个电动机轴可以做寻原点运动，也可以做速度运动，也可以做定长运动以及电子齿轮、电子凸轮等。因此，我们就可按电动机轴运动类型划分，假设在这个案例中，这个电动机轴 AxisInterfaceType 涵盖 3 种类型运动，即寻原点 Home；速度运动 Velocity，及定长运动 Position。我们创建的结构体名为 AxisInterfaceType，如图 6-17 所示。



图 6-17　创建一个名为 AxisInterfaceType 的结构变量

打开后的编辑框和设备显示如图 6-18 所示。



图 6-18　打开编辑结构变量

对结构体进行编辑，把元素寻原点 Home、速度运动 Velocity 及定长运动 Position 加入结构体中，如图 6-19 所示。

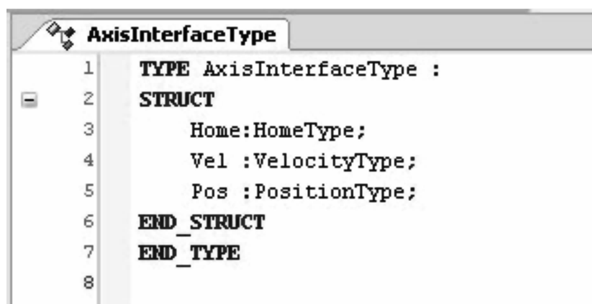


图 6-19 编辑结构体内元素组成

同理，再分别建立 HomeType、VelocityType、PositionType 的结构变量如下。在编辑 HomeType 结构变量时，我们看一下，我们在这个结构体下都需要什么元素。

寻原点功能块如图 6-20 所示。

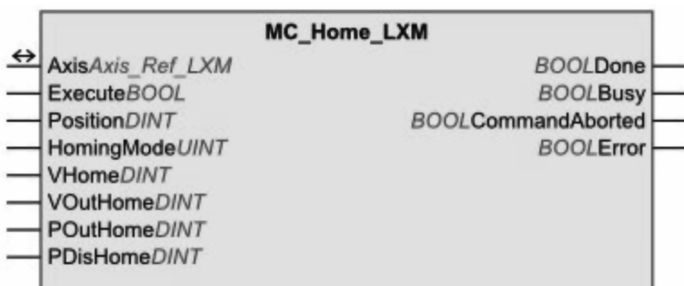


图 6-20 寻原点功能块

这样，我们知道这个功能块的输入和输出变量定义如下。

INPUT

Axis：Axis _ Ref _ LXM；

Execute：BOOL；

Position：DINT；

HomingMode：UINT；

VHome：DINT；

VOutHome：DINT；

POutHome：DINT；

PDisHome：DINT；

OUTPUT

Done：BOOL；

Busy：BOOL；

CommandAborted：BOOL；

Error：BOOL；

把变量复制到寻原点结构体下，如图 6-21 所示。

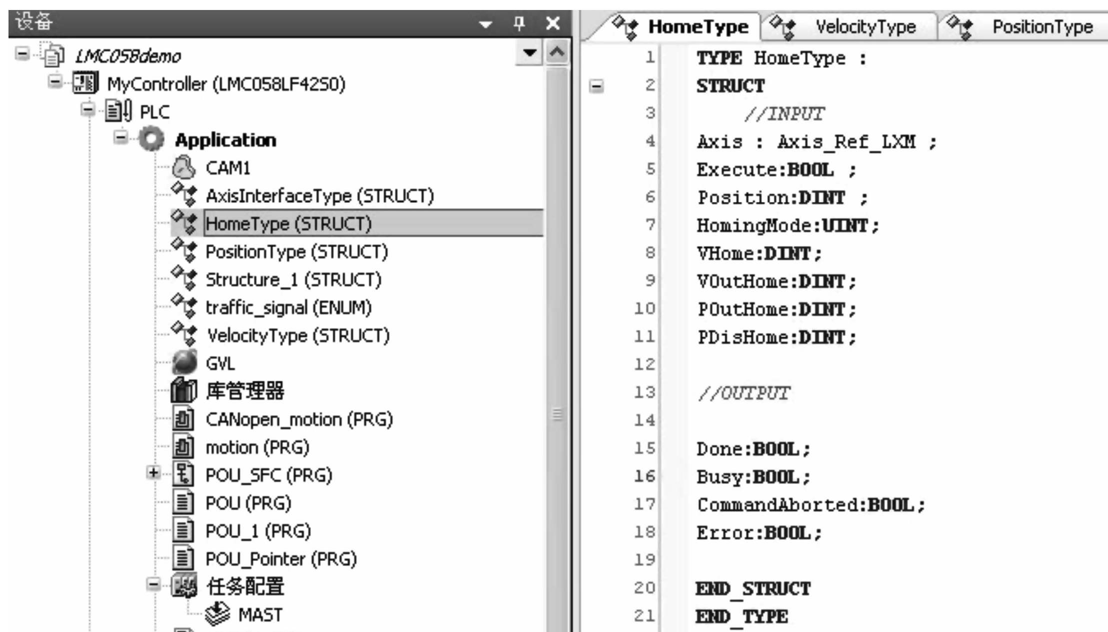


图 6-21　编辑寻原点 HomeType 结构变量下包含的元素

速度运动功能块如图 6-22 所示。

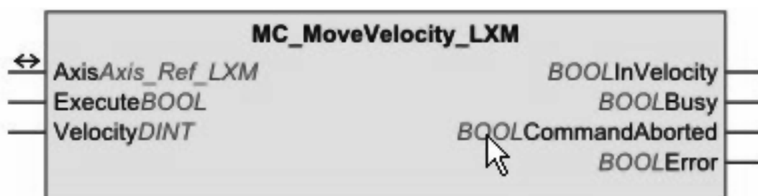


图 6-22　速度运动功能块

这样，我们知道这个功能块的输入和输出变量定义如下。

INPUT

Axis：Axis _ Ref _ LXM；

Execute：BOOL；

Velocity：DINT；

OUTPUT

InVelocity：BOOL；

Busy：BOOL；

CommandAborted：BOOL；

Error：BOOL；

把变量复制到速度运动结构体下，如图 6-23 所示。

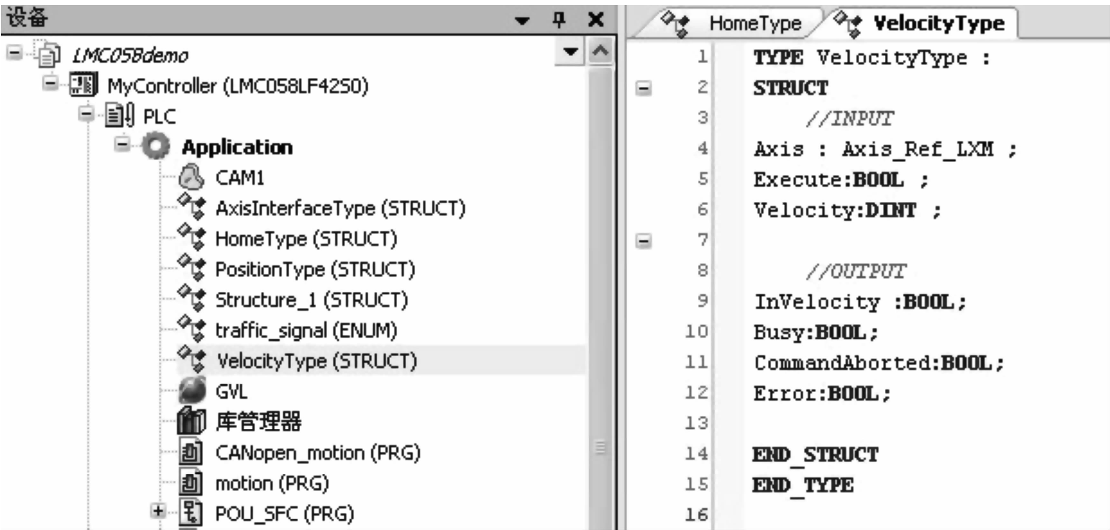


图 6-23 编辑速度运动 VelocityType 结构变量下包含的元素

定长运动功能块如图 6-24 所示。



图 6-24 定长运动功能块

这样，我们知道这个功能块的输入和输出变量定义如下。

//INPUT

Axis：Axis _ Ref _ LXM；

Execute：BOOL；

Distance：DINT；

Velocity：DINT；

//OUTPUT

Done：BOOL；

Busy：BOOL；

CommandAborted：BOOL；

Error：BOOL；

把变量复制到定长运动结构体下，如图 6-25 所示。

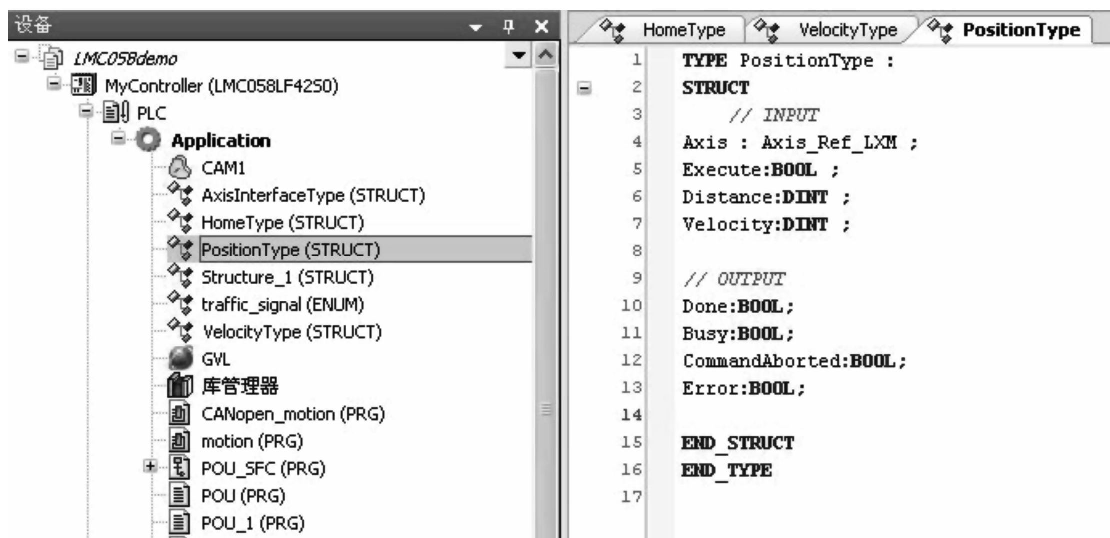


图 6-25　编辑定长运动 PositionType 结构变量下包含的元素

这样一来，当要编辑一个运动控制程序时，在变量声明区，我们定义一个轴控变量 d3 _ AxisInteface 时，把它定义为结构体变量 AxisInterfaceType，如图 6-26 所示。

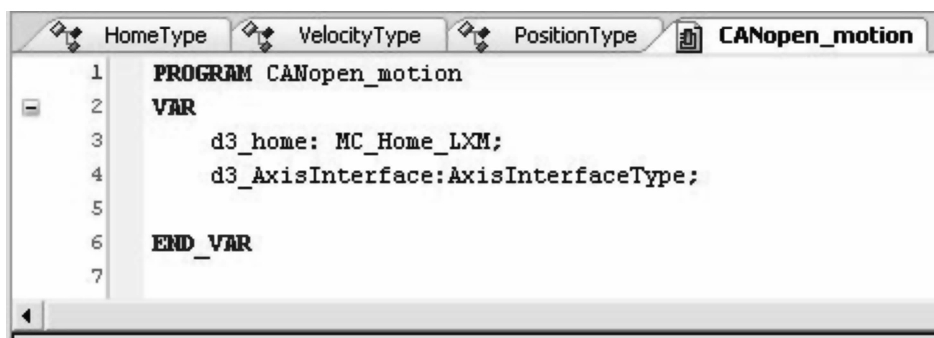


图 6-26　声明一个变量为已编辑完的结构变量

当运动模式为寻原点时，在变量“.”后缀选择 Home，如图 6-27 所示。

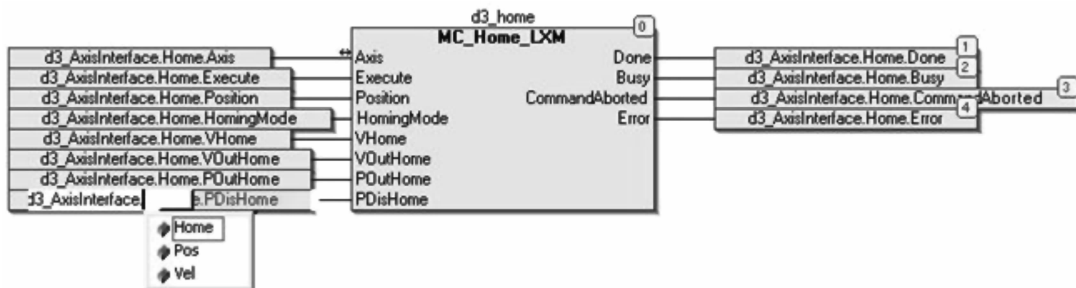


图 6-27　结构变量的首层导出

在 Home “.” 后缀出现更多变量元素，选择对应的即可，如图 6-28 所示。

最后，我们规范定义了所有变量，如图 6-29 所示。

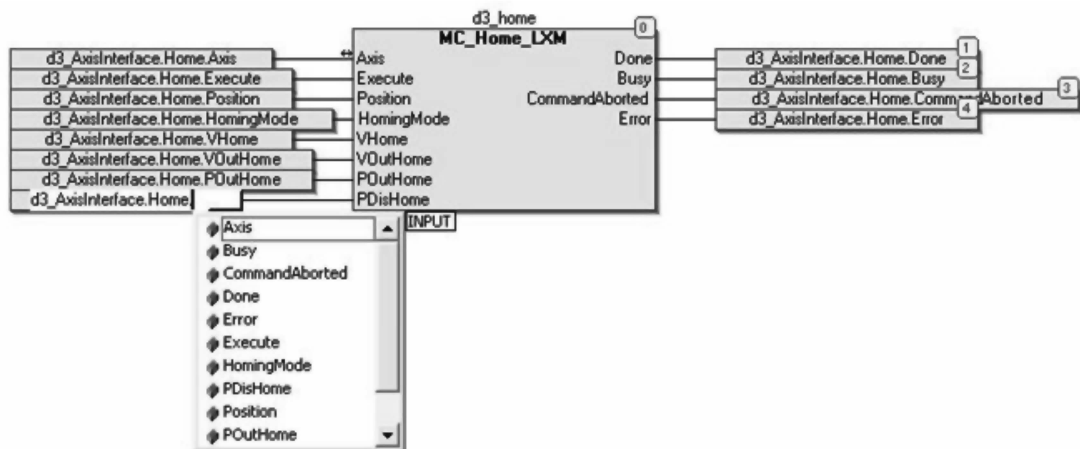


图 6-28　结构变量的第二层导出

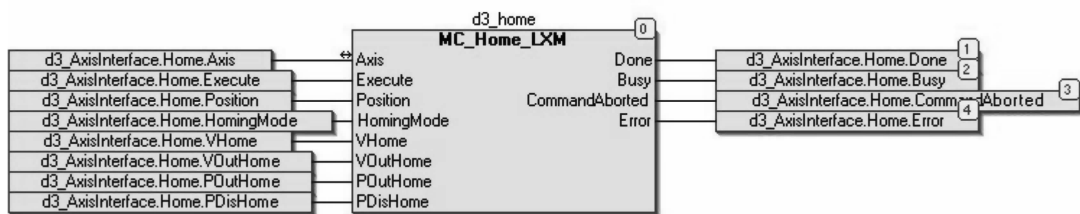


图 6-29　清晰的寻原点变量

并且，我们可以在程序中给这些变量赋值。比如给轴 Axis 赋值 d3 _ AxisInterface. Home. Axis：= d3，即运动轴为 d3，是你在硬件组态时配置好的，如图 6-30 所示。

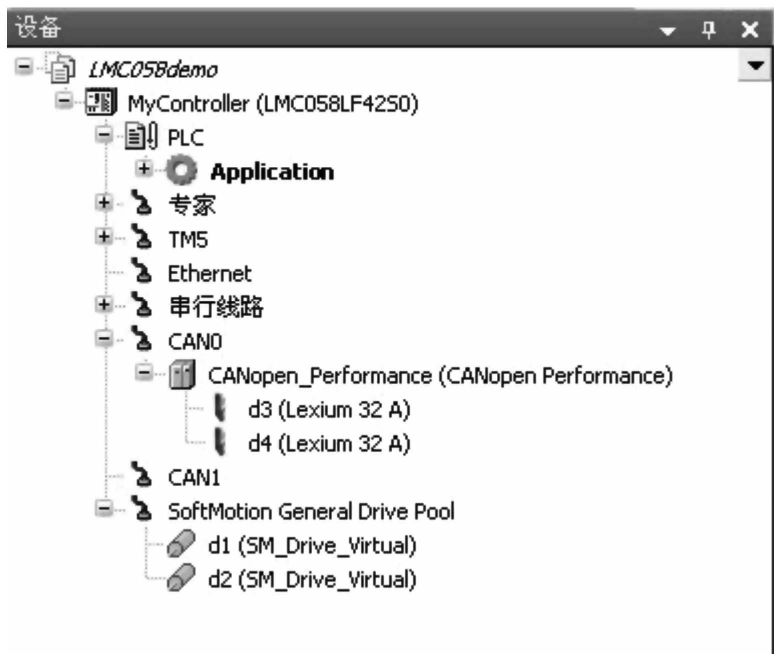


图 6-30　硬件组态好的电机轴

当你又加一个轴 d4 做寻原点运动时，你只需要在变量声明区把变量的声明再复制一次，然后把 d3 _替换为 d4 _，如图 6-31 所示。

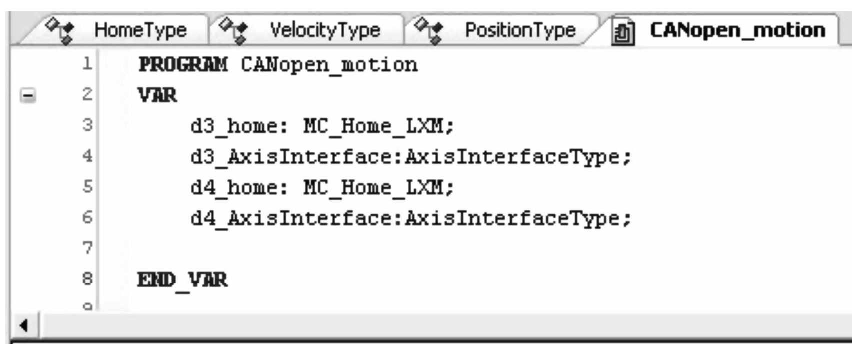


图 6-31　增加一个运动轴

在程序编辑区，复制图 6-29 中的程序一次，然后粘贴在程序编辑区，如图 6-32 所示。

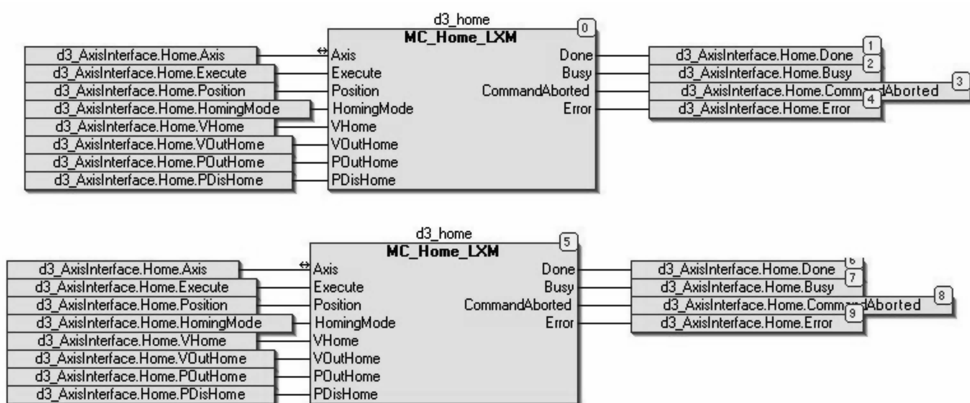


图 6-32　复制粘贴后的程序

然后利用编辑菜单中的查找 & 替换，如图 6-33、图 6-34 所示，把复制程序块中的 d3 _替换为 d4 _。



图 6-33　编辑菜单下的查找 & 替换

点击替换，则替换一步一步进行，直到把复制的功能块中的 d3 _替换完毕，如图 6-35 所示。这里要注意：不能点击全部替换。

然后，在赋值程序中对新变量也进行复制，替换即可。例如：复制一条命令 d3 _ AxisInterface. Home. Axis：= d3；把 d3 替换为 d4，d4 _ AxisInterface. Home. Axis：= d4。当运动模式为速度运动时，调入速度运动功能块，在变量声明区增加一条对功能块名 d3 _ velocity 的声明，如图 6-36 所示。

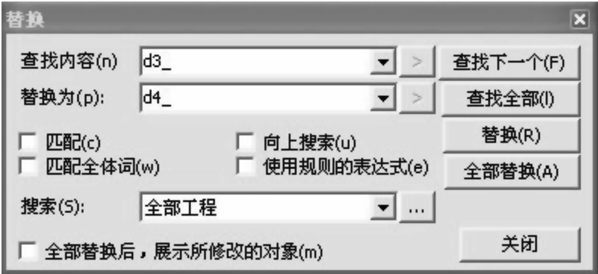


图 6-34　字符替换

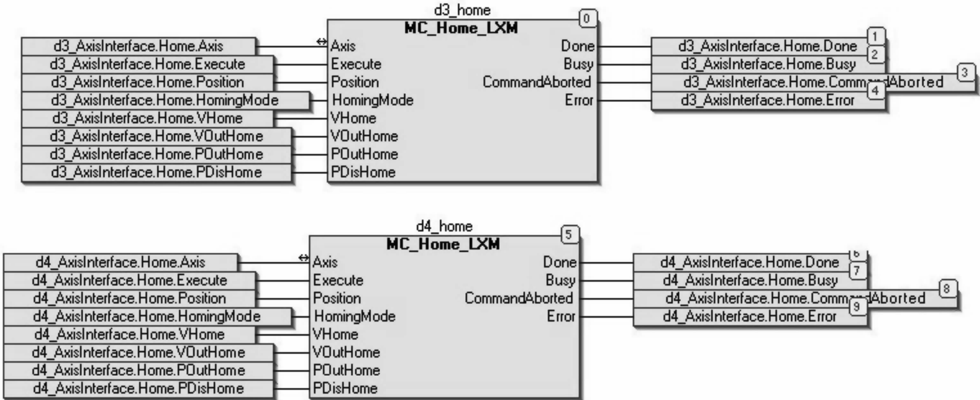


图 6-35　编辑好的程序

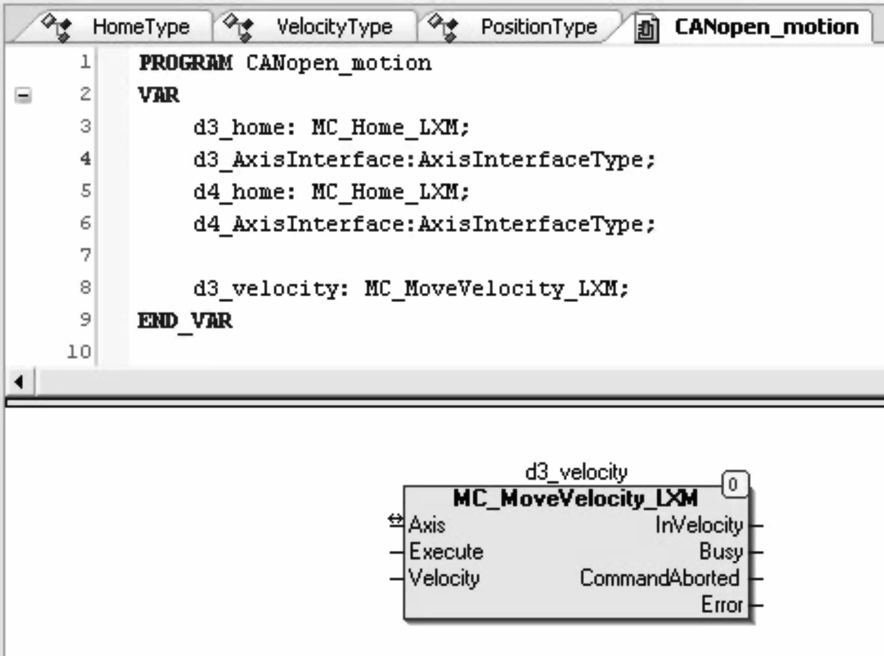


图 6-36　新增功能块及声明

这时在变量“.”的后缀选择 Vel，在 Vel“.”的后缀选择相对应变量，如图 6-37、图 6-38 所示。

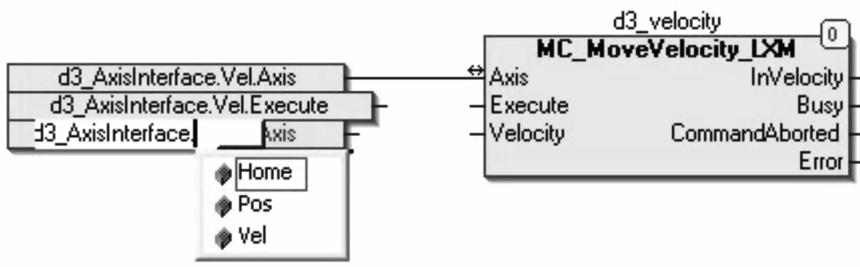


图 6-37　打开首层变量选择

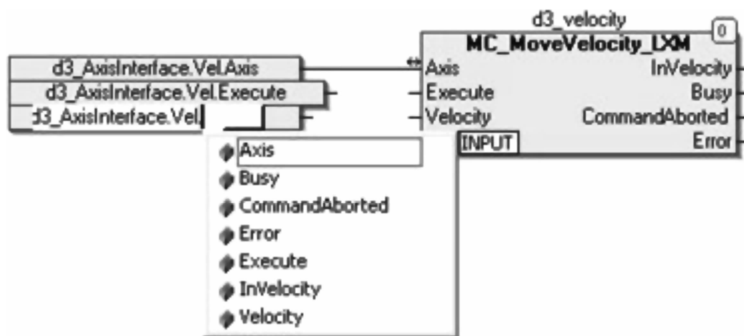


图 6-38　打开第二层变量选择

最后，我们规范定义了所有变量，如图 6-39 所示。

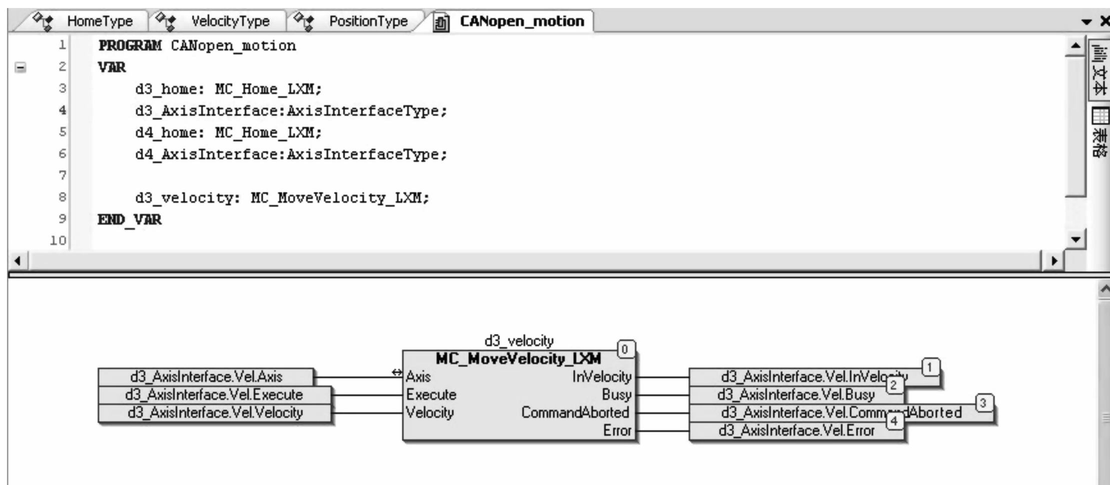


图 6-39　对速度运动的程序编辑

并且，我们可以在程序中给这些变量赋值。比如给轴 Axis 赋值 d3 _ AxisInterface. Vel. Axis：= d3，即运动轴为 d3，是你在硬件组态时配置好的。

与编辑寻原点功能块一样，我们也可以通过复制，粘贴，再替换，编辑出更多轴的速度运动。也可以用同样的方法，编辑出更多轴的定长运动。从这个案例可以看到结构体变量对编程和程序带来的益处。也就是说：结构体这种编程架构，不仅对编程技巧有所研究，而且对程序

开发的未来也包含有哲学方面的思考。它使每次编程都建立在已有的经验之上，不断构筑向上发展的大厦，使之日趋完善。而不是结构体的编程在开发新项目的时候总是推倒重来，总是在失去原有的经验和成熟工艺的情况下，使每一次的开发都成为新的探索。这无疑增加了风险和时间。而在程序设计和规划上就像一个好的作家，他不仅是熟识一种语言，更是能把这种语言组织成好的文章。不仅能使自己看懂，更重要的是使别人看懂。程序员也是一样。

6.3.3 枚举变量（Enumeration）

枚举是由很多字符串常量组成的用户定义数据类型。这些常量称为枚举值。即使在POU之内声明枚举值，枚举值仍然可以在整个工程范围内被识别。用户可以通过添加对象对话框建立“DUT”对象来创建一个枚举。

注意：与 CoDeSys V2.3 不同的是，必须使用“TYPE”才能进行局部枚举声明。

语法结构：

TYPE <标识符>:(<Enum_0>,<Enum_1>,…,<Enum_n>) <数据类型>;

END_TYPE

<标识符>变量可以代表枚举值 <Enum_…>并被初始化为第一个值。这些值与所有的数字类型兼容，也就是说可以像使用整形变量一样对其进行操作。您也可以把数 x 赋给这个值。如果在声明中枚举值没有被初始化为指定的值，那么将从 0 开始递增依次进行初始化。请确保初始化时的初始值在内部元素中递增。运行时将会检查这些值的有效性。

例如，我们建立一个枚举变量如下。

1）添加数据单元类型 DUT，如图 6-40 所示。



图 6-40 选择添加 DUT

2）命名一个枚举变量名，并选择枚举，如图 6-41 所示。



图 6-41　选择枚举

3）打开枚举编辑，如图 6-42 所示。



图 6-42　打开枚举编辑

4）编辑枚举变量下的元素，如图 6-43 所示。

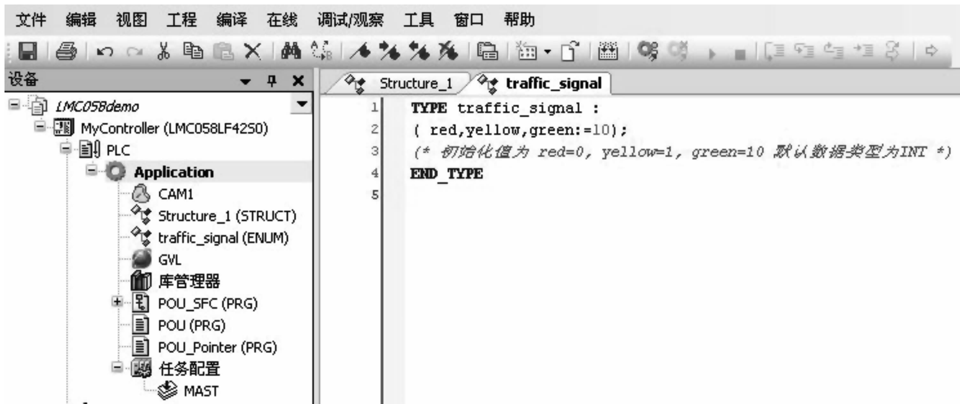


图 6-43　编辑枚举值

5）在程序中声明一个变量为一个枚举体，并在编程区调用这个枚举体变量，如图 6-44 所示。

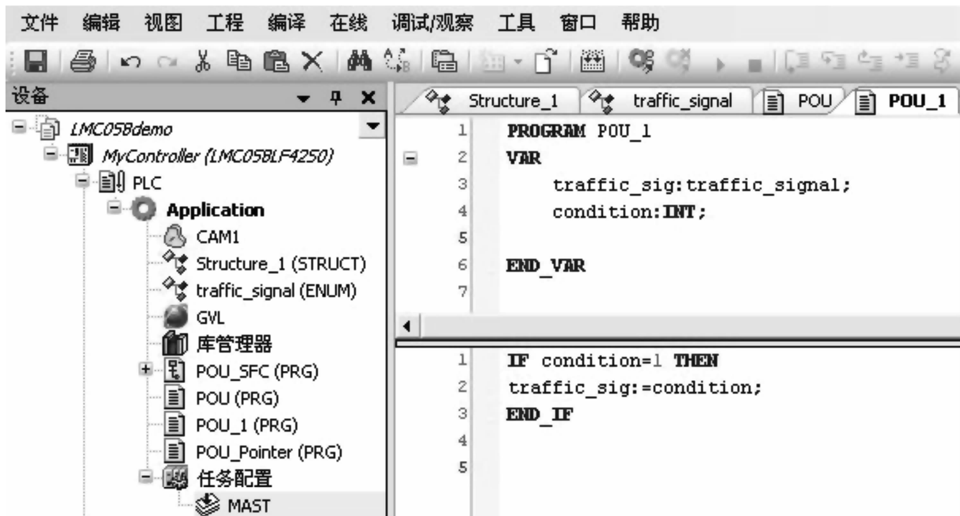


图 6-44　在程序中调用枚举变量

6）执行程序仿真，当 condition 为“0”时，条件不满足，traffic _ sig 保持初始值，即“0”，如图 6-45 所示。

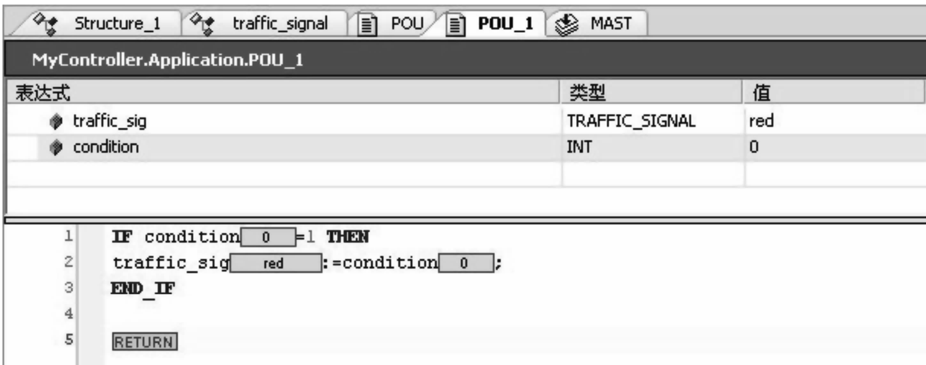


图 6-45　仿真程序（一）

当 condition 为“1”时，条件满足，traffic _ sig 被赋值为“1”，traffic _ sig 为 yellow，如图 6-46 所示。

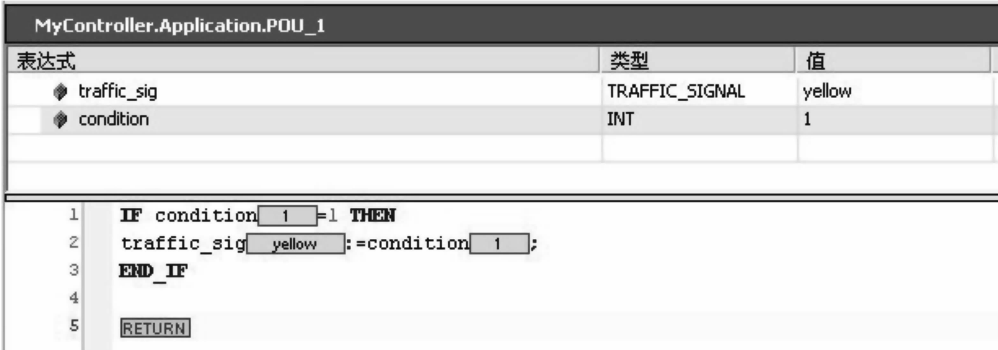


图 6-46　仿真程序（二）

枚举类型的名称可以（作为范围操作符）用来确定需要访问的枚举常量。因此，您可以在不同枚举中使用相同的常量。

例如：

定义两个枚举：

TYPE COLORS _ 1：（red，blue）；

END _ TYPE

TYPE COLORS _ 2：（green，blue，yellow）；

END _ TYPE

在 POU 中使用枚举值 Blue：

声明：

colorvar1：COLORS _ 1；

colorvar2：COLORS _ 2；

实现：

可以这样使用：

colorvar1：= colors _ 1. blue；

colorvar2：= colors _ 2. blue；

不能这样使用：

colorvar1：= blue；

colorvar2：= blue；

还可以明确指定枚举所指向的数据类型（默认类型是 INT）。

例如：

需要将枚举 BigEnum 的数据类型指定为 DINT：

TYPE BigEnum：（yellow，blue，green：= 16#8000）DINT；

END _ TYPE

6.4 带有物理地址的变量

1. 控制器变量的物理地址

在控制器中，经常使用的符号如下，它们代表了使用的功能和地址分布状态。PLC 地址定义及分布如图 6-47 所示。

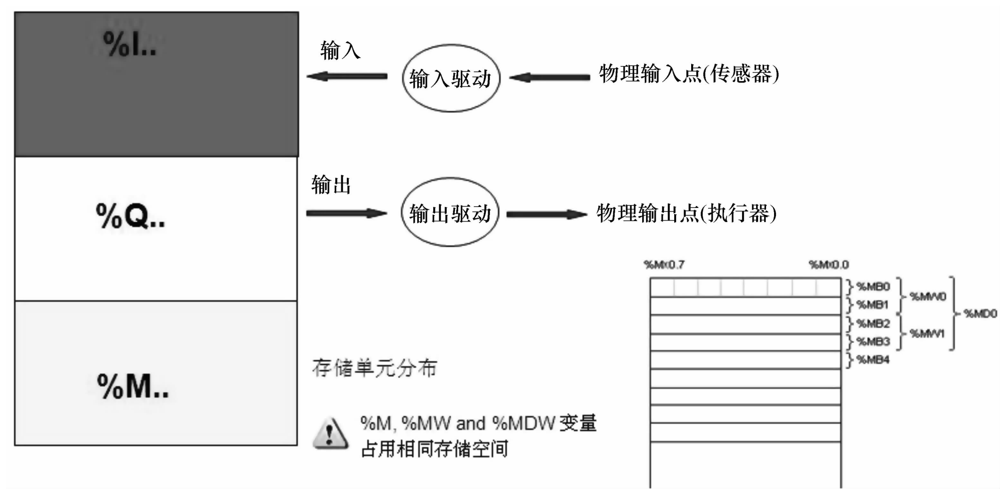


图 6-47 PLC 地址定义及分布

地址语法表示

- 百分号‘%’
- 功能区-前缀
  - I 输入
  - Q 输出
  - M 存储单元区
- 容量
  - X 位
  - B 字节（8 位）
  - W 字（16 位）
  - D 双字（32 位）

例如：

% IW215 = 输入字 215

% QX3.1 = 输出字节 3 的第一位

% MD48 = 双字 48 存储单元

2. I/O 映射

设备配置对话框（设备编辑器）被命名为“ <设备类型> I/O 映射”（例如，“DP-模块 I/O 映射”）。它用来在工程变量上映射 PLC，也就是为应用程序的变量分配设备输入、输出地址和存储地址。

应用目前操作的 I/O 被定义在 PLC 设置对话框中，如图 6-48 所示。

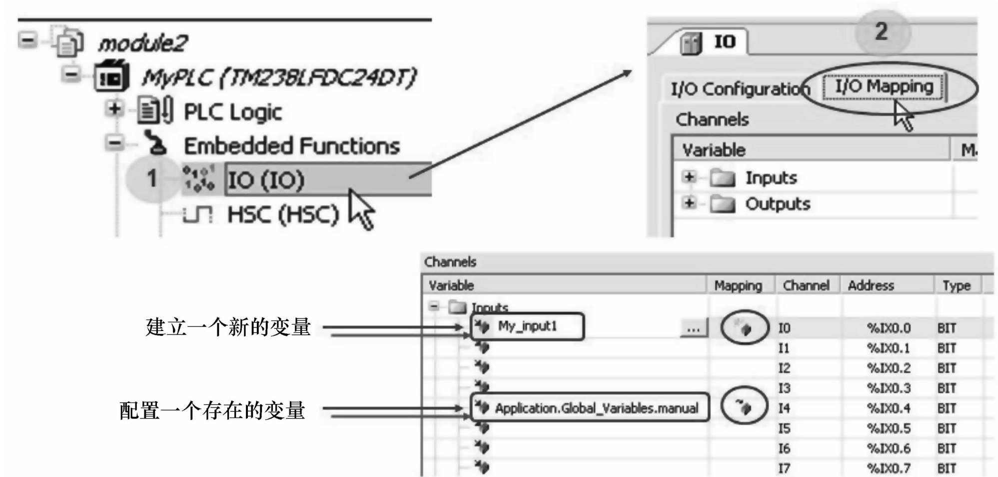


图 6-48　I/O 映射配置

你可以在 SoMachine 在线帮助文件中，参阅下列内容：

I/O 映射信息；

使用 I/O 映射对话框；

强制 I/O 隐式变量。

3. 关键字“AT”

除了使用 I/O 映射对话框，通过“AT 声明”也可以为一个变量分配一个地址。例如：

My _ input7 AT %IX0. 7：BOOL；

Trigger _1 AT %MX2. 2：BOOL；

这些变量声明可以在一个程序块中或全局变量列表中实现。

第 7 章　在线配置和组态任务

7.1　网关组态

网关（Gateway）的作用：网关给了应用者使用计算机进入控制器网络的功能，如图 7-1 所示。通常接入的是本地主设备并且可以扫描接入主设备的所有其他设备，如图 7-2 所示。

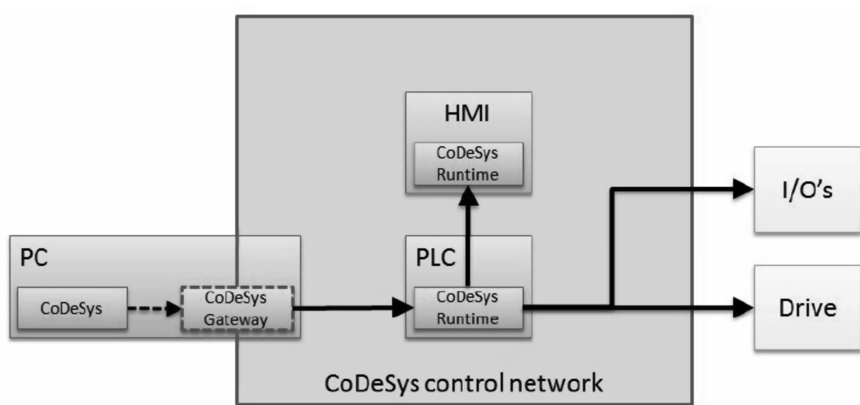


图 7-1　网关的作用

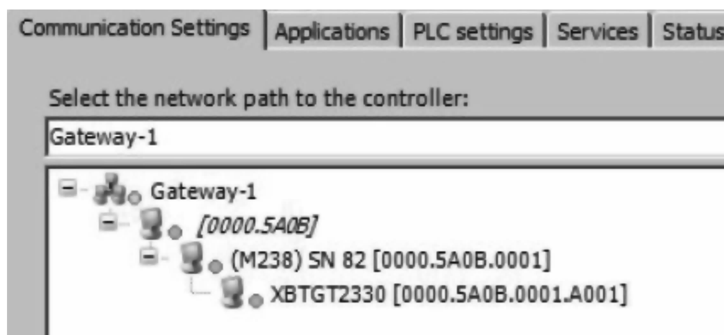


图 7-2　扫描到主设备和连接到主设备的其他设备

网关的路由：如图 7-3 所示，接入计算机可以有两种界面：一个是 USB，另一个是以太网。当计算机扫描到设备后，地址被显示在方括号内，如图 7-2 所示。地址也会因路由的拓扑结构改变而变化。网络站点名也可以作为地址。

从图 7-4 可看到，站点名和控制器名不一样。

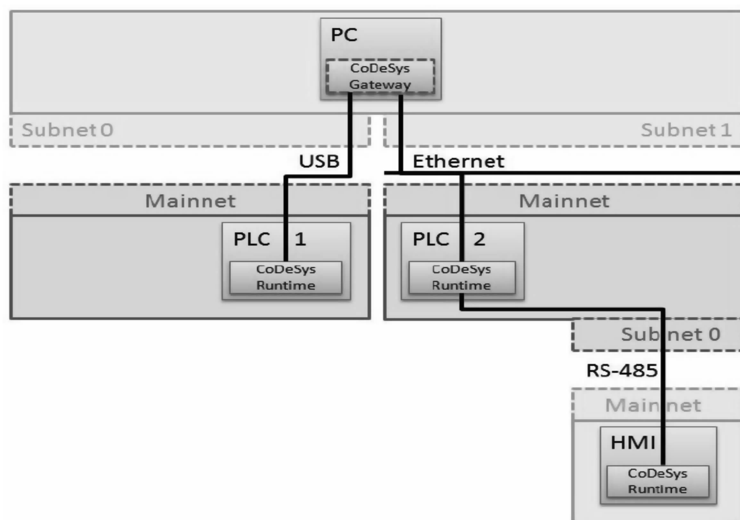


图 7-3　网关的路由

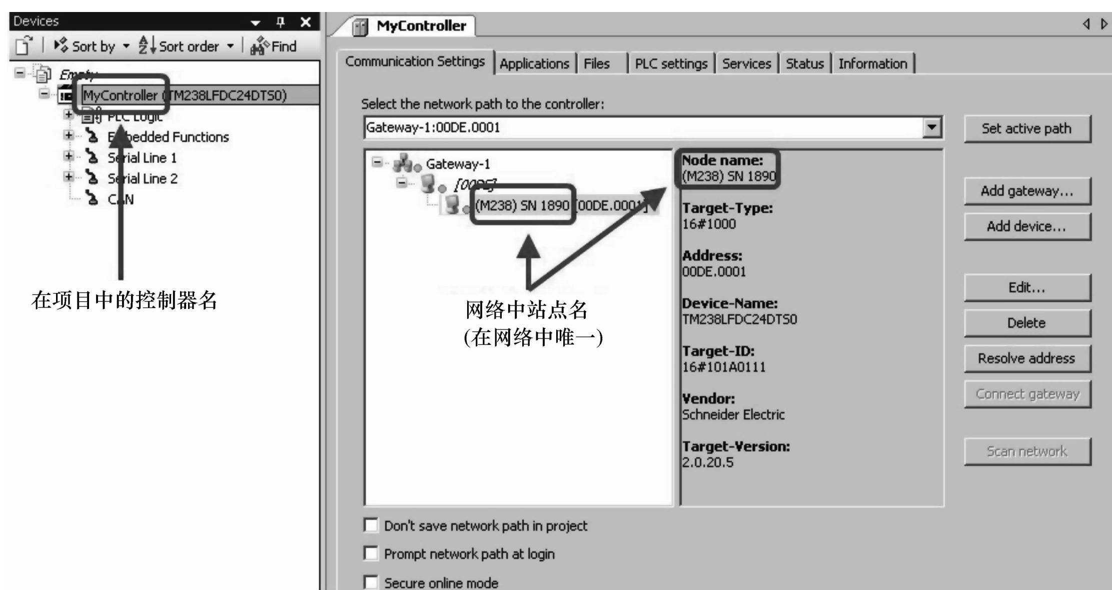


图 7-4　网络中的站点名

7.2　应用操作

1. 激活当前控制器

如图 7-4 所示，当扫描到设备后，点击“设置使用路径（Set active path)”，然后按“ALT + F”，得到站点名和地址显示变黑，并标注“活动的”（ACTIVE)。当一个项目下，连接多个控制器时，在配置栏目或设备表下，点击鼠标右键，会出现“激活当前应用（Set Active Application)”，确认后，可实现对激活控制器的操作。激活控制器如图 7-5 所示。

- 一个项目下可连接多个控制器
- 只能对激活的控制器进行操作

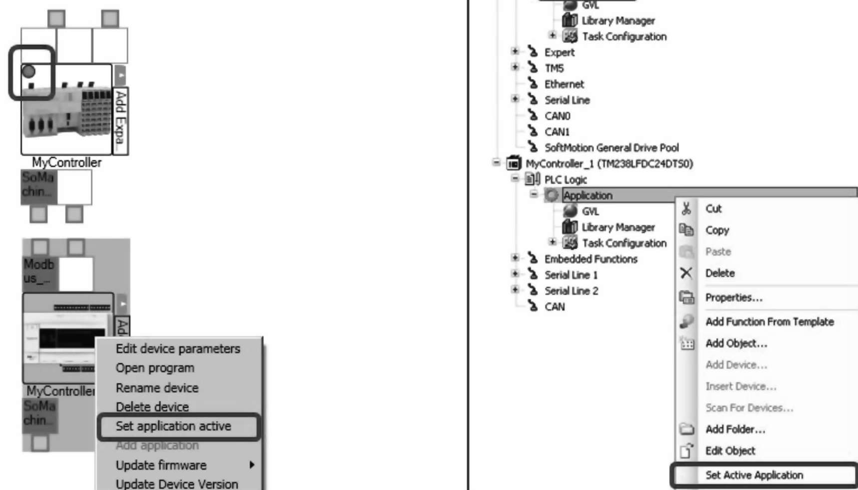


图 7-5　多控制器下激活要操作的控制器

2. 程序的编译

在程序栏目下，点击“编译”，打开编译菜单如图 7-6 所示。



图 7-6　打开编译菜单

当要编译一个项目下的多个控制器程序时，选择全部生成；当编译激活的控制器时，选择编译（快捷键“F11”）；当只编译修改的部分时，选择重新编译。离线测试可选择生成代码；完成生成代码，但不用下载，可选择生成后配置并可把文件保存到计算机中，如图 7-7 所示。

编译完的代码下载到控制器的 RAM 如图 7-8 所示。

3. 创建启动应用

- 调用生成代码到 flash；



图 7-7　保存配置文件

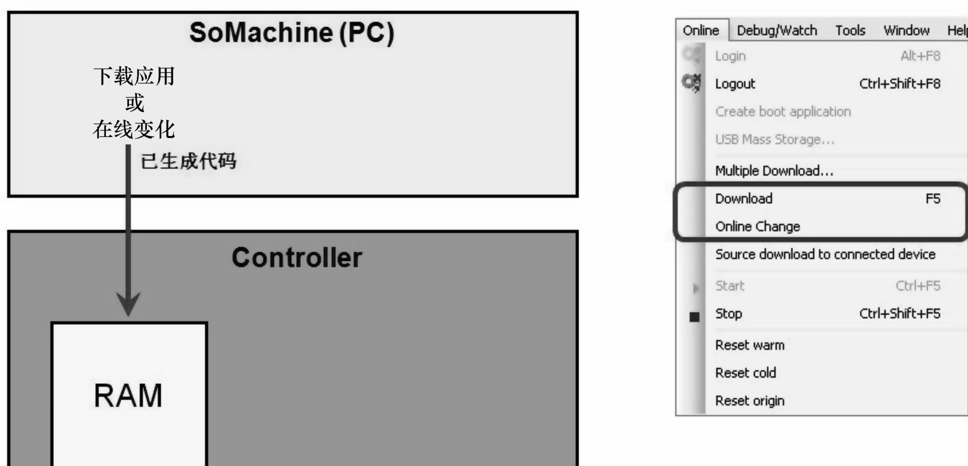


图 7-8　编译完的代码下载到控制器的 RAM

- M238 必须在停止状态；
- 上电周期；
- 程序代码从 FLASH 到 RAM。

创建启动应用如图 7-9 所示。

4. 自动创建启动应用下载

按如图 7-10 设定后，每一次应用更新后，登录控制器都会完成自动下载。

5. 启动应用的状态显示

（1）在控制器上的 LED 状态指示

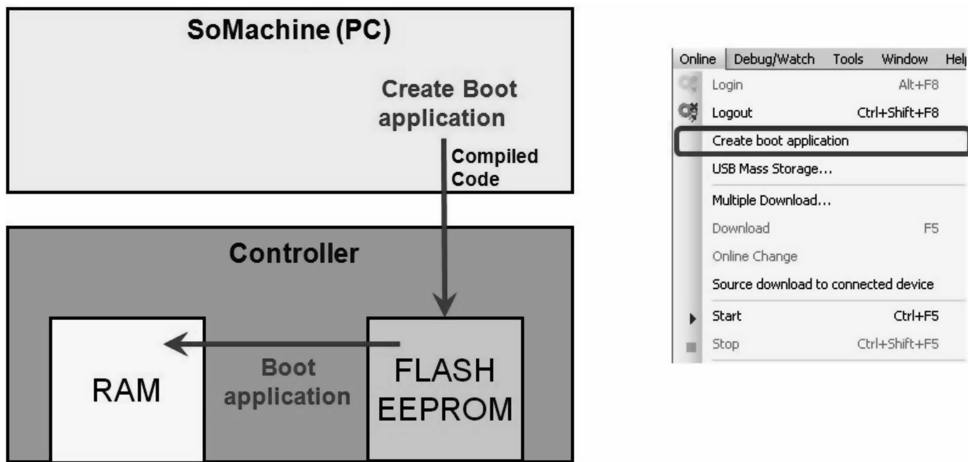


图 7-9　创建启动应用

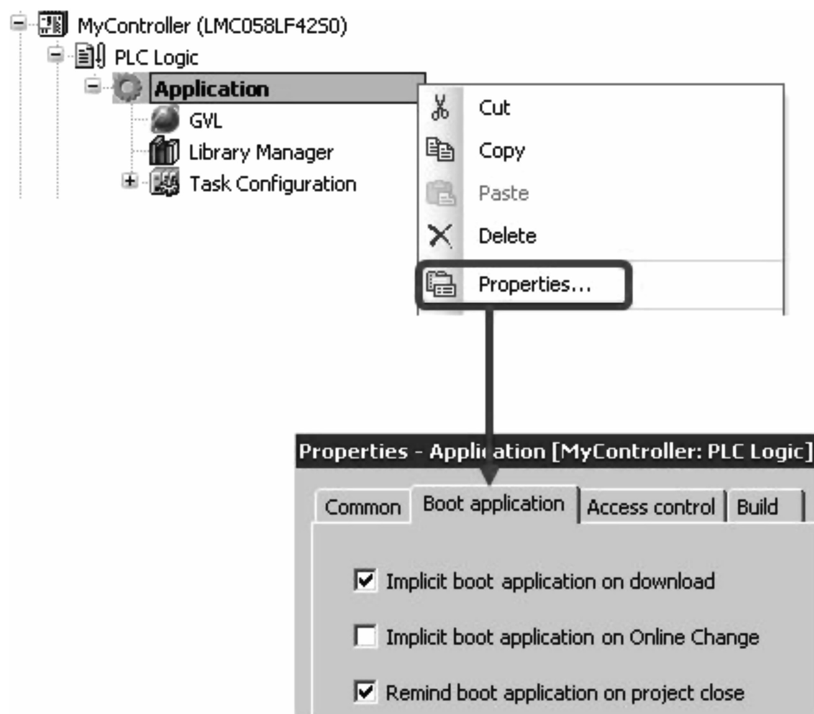


图 7-10　设定自动启动应用

- M238-ERR 灯慢闪（25% ON，75% OFF）；
- M258/LMC058-RUN 绿色（RUN）或红，绿交替闪动（STOP）。

（2）SoMachine 编程软件上的信息警示

信息警示如图 7-11 所示。

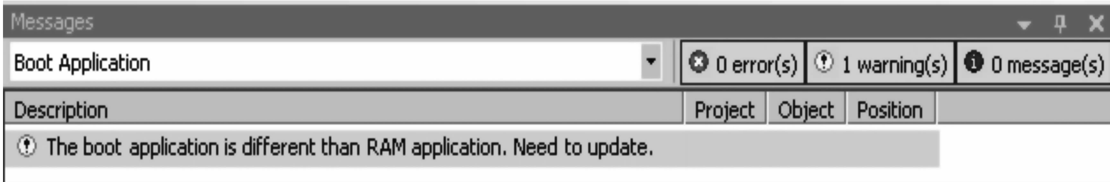


图 7-11　信息警示

系统变量 PLC _ R 如图 7-12 所示。

Watch 1

Expression	Comment	Type	Value
MyController_1.Application.PLC_R.i_wBootProjectStatus		PLC_R_BOOT_PROJECT_STATUS	PLC_R_BOOT_PROJECT_STATUS.PLC_R_DIFFERENT_BOOT_PROJECT

图 7-12　系统变量 PLC _ R

6. 源代码的下载和上传

源代码存储在控制器的路径为：/usr/App。

（1）下载到控制器

控制器离线时，可采用打开文件菜单，选择下载源代码，如图 7-13 所示。



图 7-13　离线下载

控制器在线时，可采用打开在线菜单，选择下载源代码到连接设备上，如图 7-14 所示。

（2）上传到计算机

控制器离线时，可采用打开文件菜单，选择源上传，如图 7-15 所示。

控制器在线时，可打开控制器栏目下的文件条目，从右侧控制器中选择源代码程序，上传到计算机中。

7. 自动源代码下载

如图 7-16 所示，打开工程菜单，点击工程设置，进入工程设置菜单。

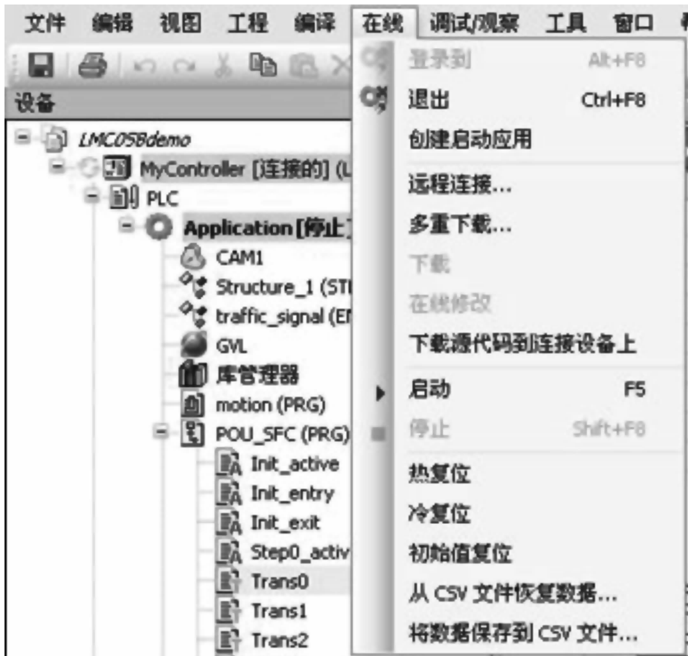


图 7-14　在线下载

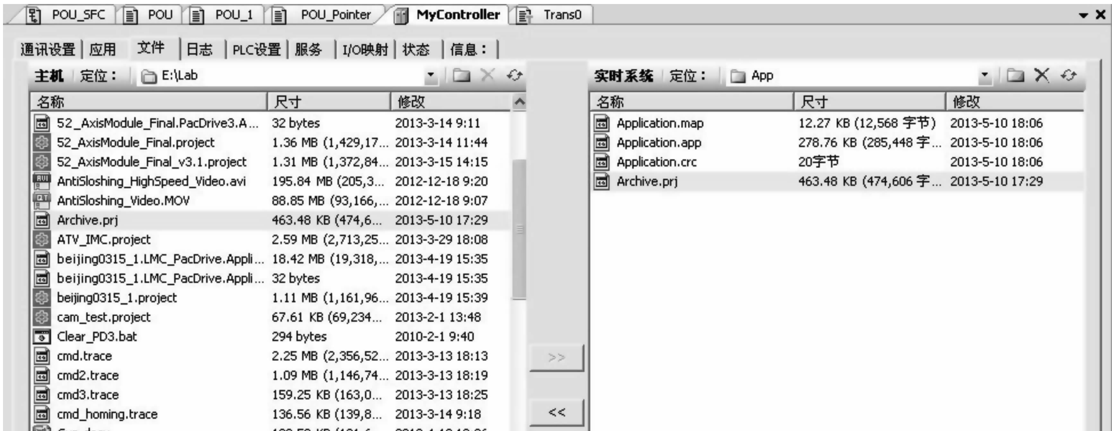


图 7-15　在线上传

组态工程设置如图 7-17 所示。

在组态工程设置框，点击附加文件，会弹出附加文件框，如图 7-18 所示，选择要下载的文件即可。设置完成后，每当登录控制器时，相关文件就会自动下载。

8. 文件的后缀表示

- SoMachine 项目文件　　<文件名>. project
- SoMachine 项目备份文件　　<文件名>. backup
- SoMachine 压缩文件　　<文件名>. projectarchive
- 使用设定　　<文件名>-USER. opt（如，设置观察的内容）
- 应用码 &ID　　<文件名><设备><ID>. compileinfo



图 7-16　打开工程设置菜单



图 7-17　组态工程设置

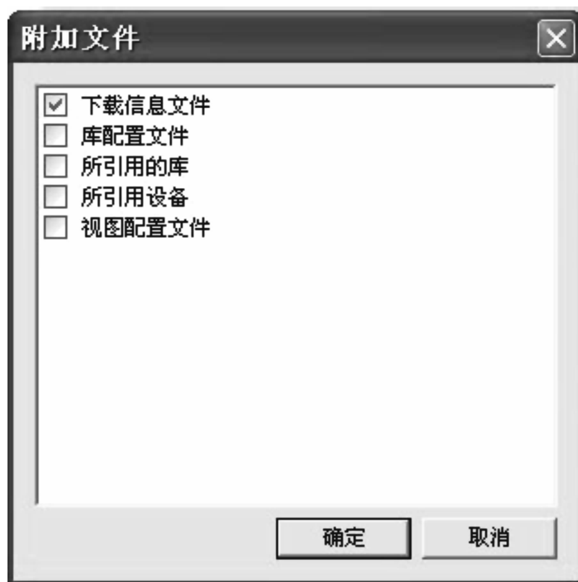


图 7-18　选择要自动下载的文件

- 启动应用　　< application >. app
　　< application >. crc
　　< application >. map
- 控制器中的源文件　　Archive. prj
- 通信组态　　Machine. cfg（界面设定，如 . IP 地址）

7.3　PLC 设置和管理

在设备栏目下，双击“MyController”，打开了控制器的设置和管理界面，如图 7-19 所示。

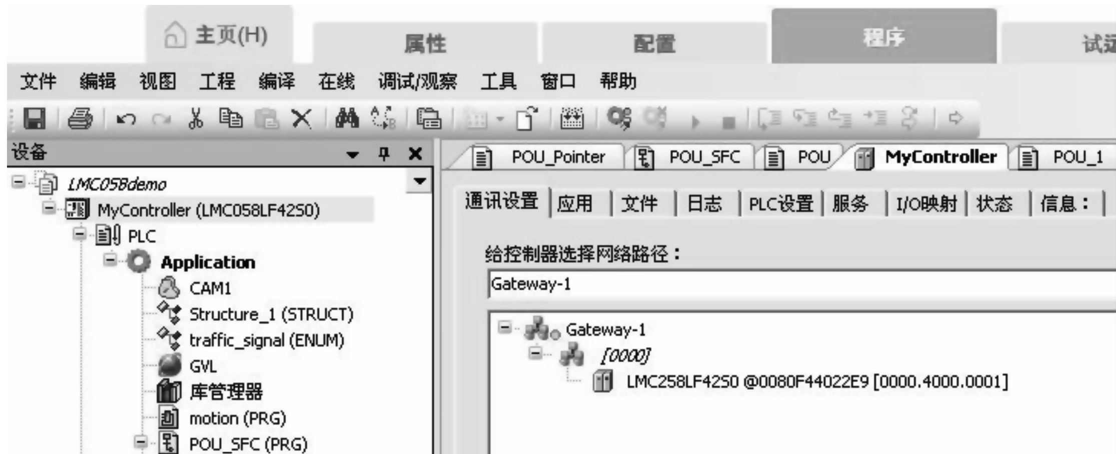


图 7-19　打开控制器的设置和管理界面

在这个界面，我们可以看到控制器的管理和设置条目。例如，点击“通讯设置”，在这个条目下，我们可以创建一个网关，扫描连接在控制网络上的控制器和设备。点击“应用”，我们打开了一个在控制器上显示已调入的应用框。在这个界面，我们可以看到运行的程序，也可以删除一个程序，如图 7-20 所示。

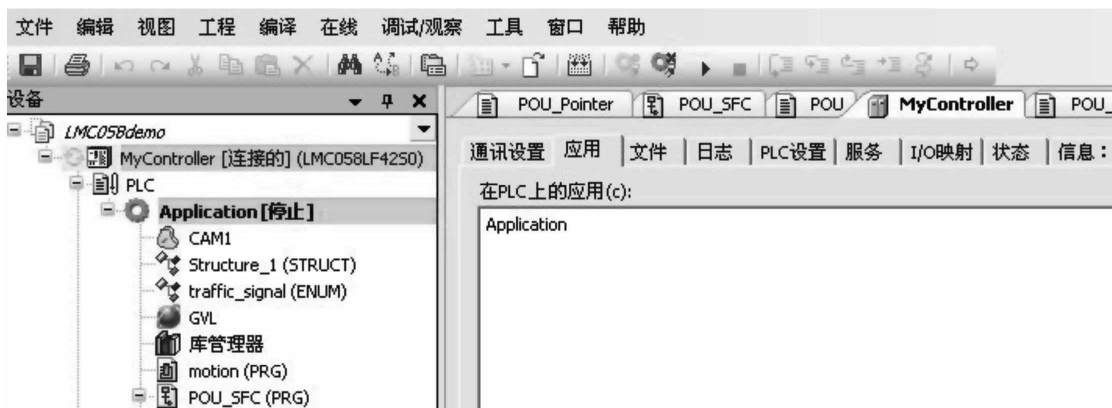


图 7-20　打开应用

点击“文件”，我们就可以在登录控制器后，实现计算机（左边）和控制器（右边）之间的文件交换。例如，我们可以把编好的 CNC G 代码传送到控制器里，如图 7-21 所示。

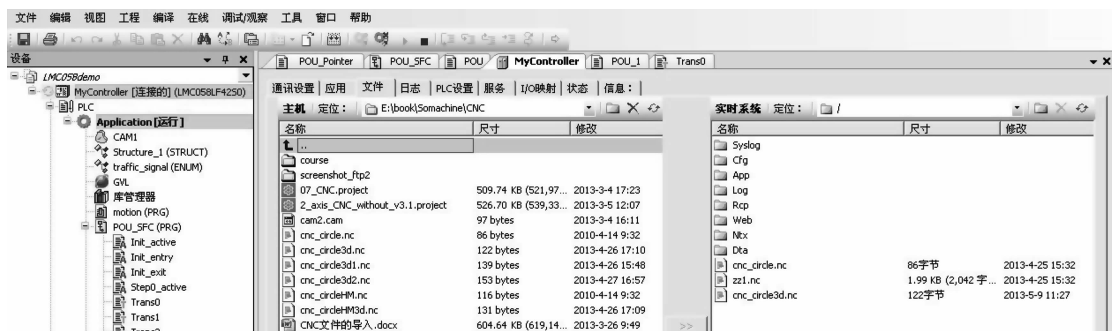


图 7-21　打开文件条目，实现文件交换

点击“PLC 设置”，可以对 PLC 的扫描任务和输入/输出对应的应用进行配置，如图 7-22 所示。



图 7-22　对 PLC 进行设置

点击“服务”，可以对控制器写入实时时钟，察看设备标识，例如：固件版本等，如图 7-23 所示。

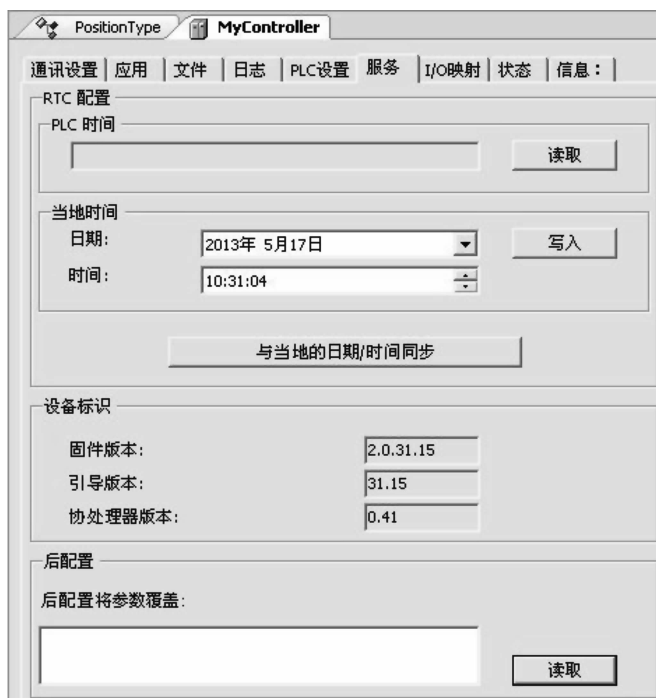


图 7-23　设置时钟

点击“状态”，可以看到控制器在线时的状态信息，如图 7-24 所示。



图 7-24 观察控制器状态

7.4 内置 I/O 的组态

当对控制器的输入组态时，也可以对输入点的参数进行配置。这不同于对控制器上的输入点进行配置。例如我们在控制器上定义一个输入点用于传感器输入，但这个输入点的参数配置意味着这个点可以有些输入是延时的，用于滤波或这个点具有锁存功能或可以启动一个中断程序。在配置如图 7-25 所示的 M238 控制器时，在设备栏目，双击内置 I/O，打开如图 7-26 所示的 I/O 配置画面。



图 7-25 配置 M238 逻辑控制器内嵌 I/O

在 M238 逻辑控制器的 I/O 配置界面，我们可以看到输入点的参数定义。从图 7-26 可以看到，在默认状态，跳动过滤器（防颤过滤）是处于失效状态的，它的激活取决于输入点用于锁存或事件功能。也就是说，一个输入点的防颤过滤只有在该点定义为锁存或事件点时才能使用。过滤时间可以点击应用值的下拉菜单来选择。同样输入点的功能定义，可以点击应用值框的下拉菜单来选择。在图 7-26 中我们还注意到，只有快速点具有锁存和事件功能，而在普通点则没有锁存和事件功能。

在 M258 逻辑控制器，LMC058 运动控制器 RUN/STOP 输入的定义在专家模板进行

- 专家模板 1（DM72F0）→BLOCK0 _ I0. . BLOCK0 _ I5
- 专家模板 2（DM72F1）→BLOCK1 _ I0. . BLOCK1 _ I5

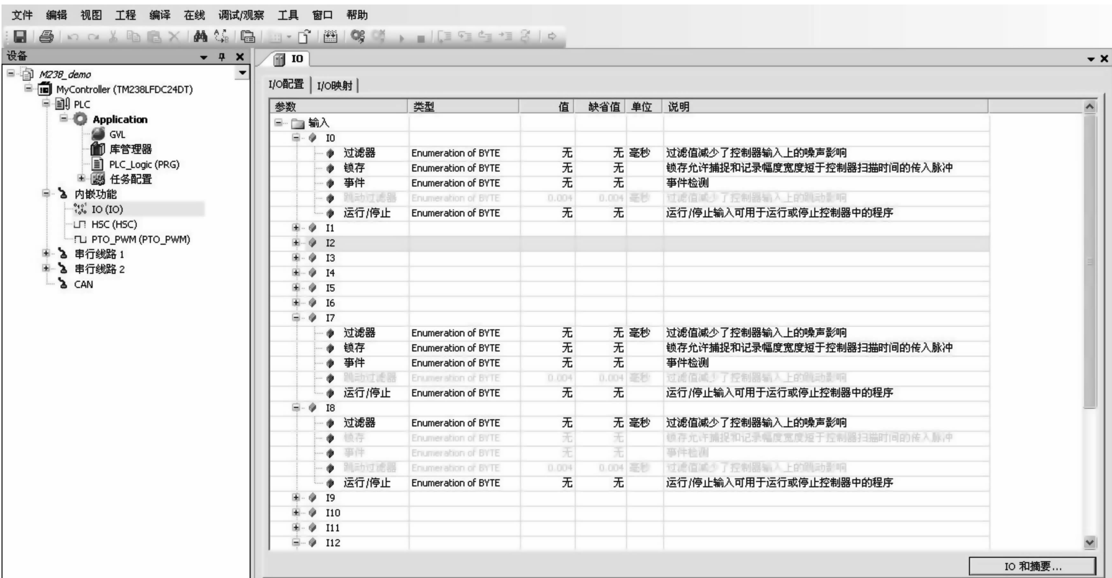


图 7-26　I/O 配置界面

双击“专家 Expert”，会打开如图 7-27 所示界面，进行 I/O 功能配置。

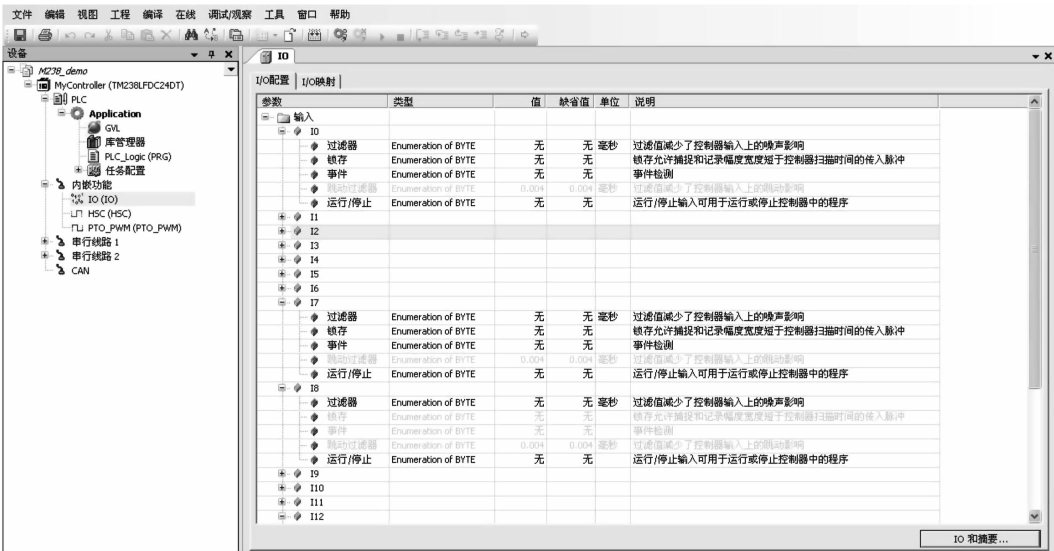


图 7-27　I/O 配置界面

例如选择参数所列出的启动/停止、报警输出、重置输出模式等功能，并在执行值参数下拉菜单中选择 I/O 配置。

在对专家模板功能的定义上（比如对锁存/事件 Latch/Event，计数器 Counter）我们可以用鼠标把标示移动到模板图标上，点击右键打开菜单，选择所要实现的功能，即添加一个功能设备。如图 7-28 所示添加一个子设备，来使这个模板实现某种功能。

- 定义输出后备状态一般为默认值，如图 7-29 所示。这样当 PLC 停止运行时，输出按

默认值输出。

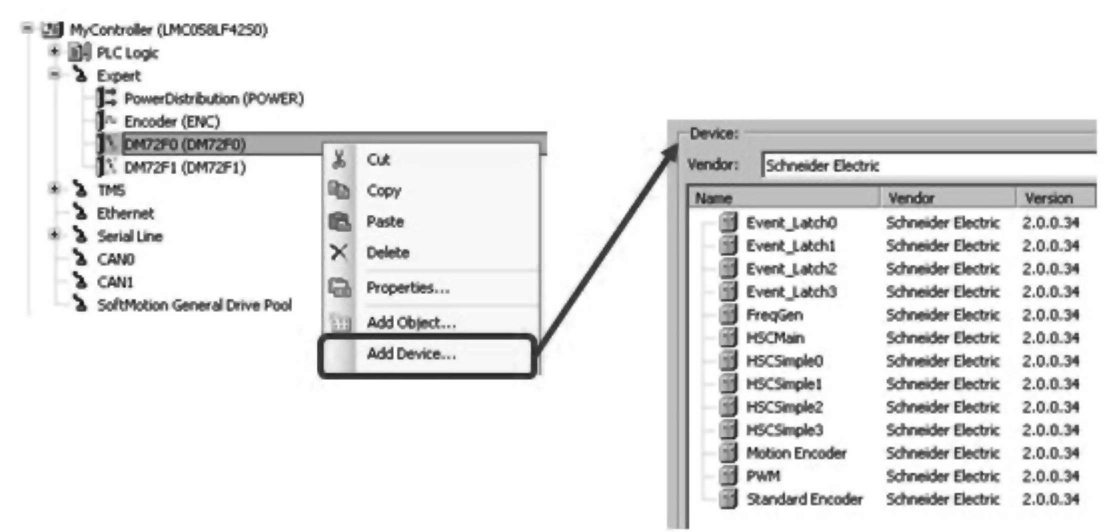


图 7-28　添加一个子设备

I/O Configuration　I/O Mapping

Channels

Variable	Mapping	Channel	Address	Type	Current Value	Default Value	Unit	Description
Inputs								
Outputs								
		QW0	%QW0	WORD				
		Q0	%QX0.0	BOOL		TRUE		Fast Output
		Q1	%QX0.1	BOOL		FALSE		Fast Output
		Q2	%QX0.2	BOOL		TRUE		Fast Output
		Q3	%QX0.3	BOOL				Fast Output
		Q4	%QX0.4	BOOL				
		Q5	%QX0.5	BOOL		TRUE		
		Q6	%QX0.6	BOOL				
		Q7	%QX0.7	BOOL				
		Q8	%QX1.0	BOOL				
		Q9	%QX1.1	BOOL				

图 7-29　配置 PLC 停止后的默认值

- 模拟输出组态。如图 7-30 所示，输出为模拟量时，可通过 I/O 配置来定义输出类型。

I/O Configuration　Expansion Bus I/O Mapping　Status　Information

Parameter	Type	Value	Default Value	Unit	Description
Outputs					
QW0					
Type	Enumeration of BYTE	0..10 V (Not used / 0..10 V / 4..20 mA)	Not used		Range mode
Scope	Enumeration of BYTE		Not used		Unit
Minimum	INT		0		Minimum value
Maximum	INT		4095		Maximum value

图 7-30　配置模拟量输出

7.5　任务组态和运行

SoMachine 控制系统的执行具有以下特点：

- 系统是多任务执行的。
- 最大可执行任务数由控制器类型决定。
- 不同的任务采用不同的执行方式。
- 程序组织单元只有放入任务中，才会被控制器执行。
- 只有程序组织单元 POU 才能加到任务栏中。功能块是不能加入到任务栏中的。

控制器扫描的顺序如图 7-31 所示。

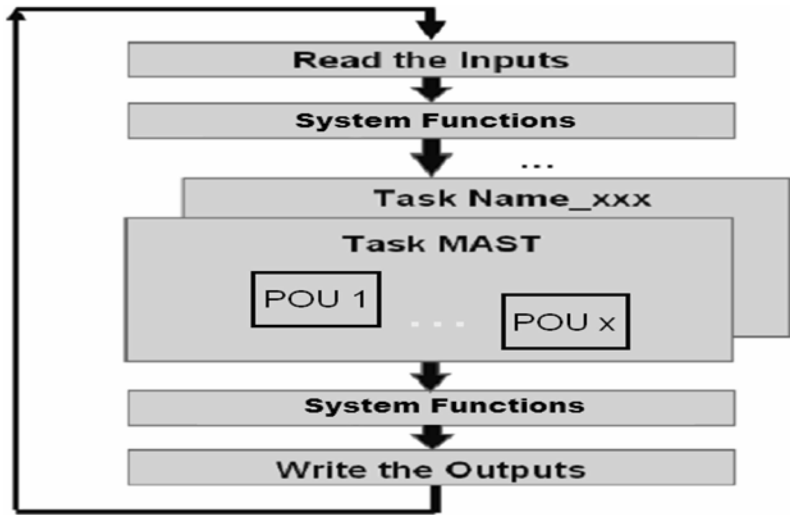


图 7-31　控制器扫描的顺序

以下是任务执行的不同方式。

1）周期执行方式。如图 7-32 所示，每一次扫描的周期一致，也就是每一次扫描输入和程序字头是规律的、定时的。这样在与外部信号的对接中，会规避竞争冒险。图中的 I. P. 是扫描内部功能。

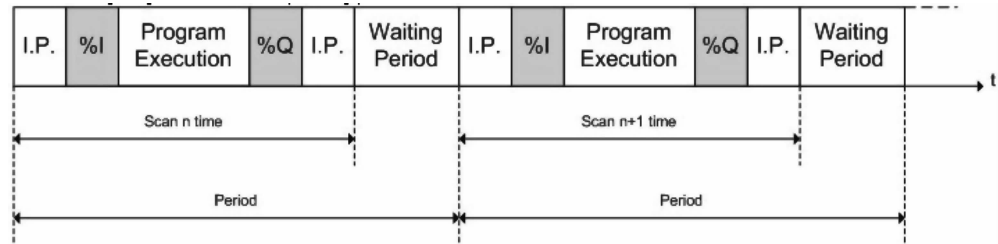


图 7-32　周期执行任务

2）事件任务。即引起执行一个中断任务。它按引起中断的方式在任务类型上分为事件（内部）和外部。当引起中断的信号来自内部的一些变量时，我们在类型栏目选择事件，并

在事件的下拉菜单中选择变量。当引起中断的信号来自控制器硬件端口，则在类型栏目选择外部的，并在外部事件栏的下拉菜单中选择端口信号。事件任务的执行如图 7-33 所示。

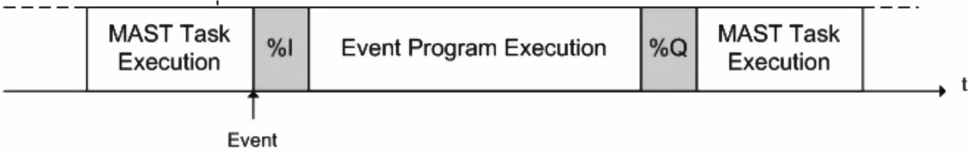


图 7-33　事件任务的执行

3）自由运行。这种扫描方式为一旦结束一个程序扫描就马上返回程序字头，开始新的一次扫描。图 7-34 中的 I. P. 是扫描内部功能。

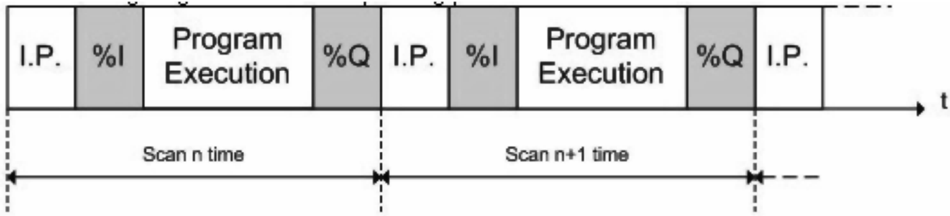


图 7-34　自由循环任务

在组态任务时，可根据不同要求，配置任务的执行方式和优先级别。图 7-35 示出了任务的优先级和程序组织单元 POU 的执行顺序。

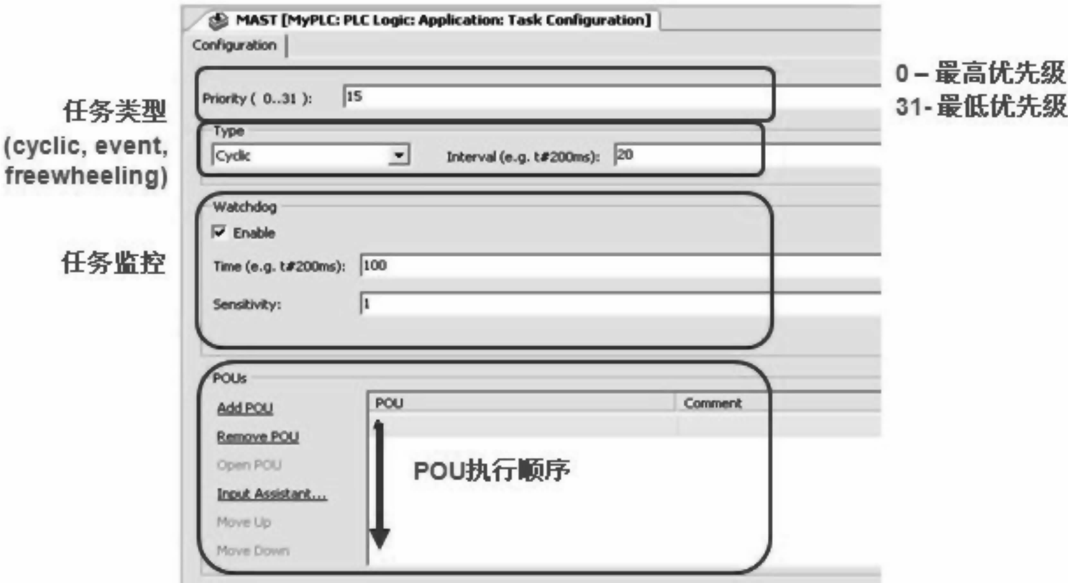


图 7-35　任务的优先级和程序组织单元 POU 的执行顺序

例如在 M238 逻辑控制器上，我们对端口 I2 点进行组态，使它具有启动事件功能，并且启动事件的信号为上升沿，如图 7-36 所示。

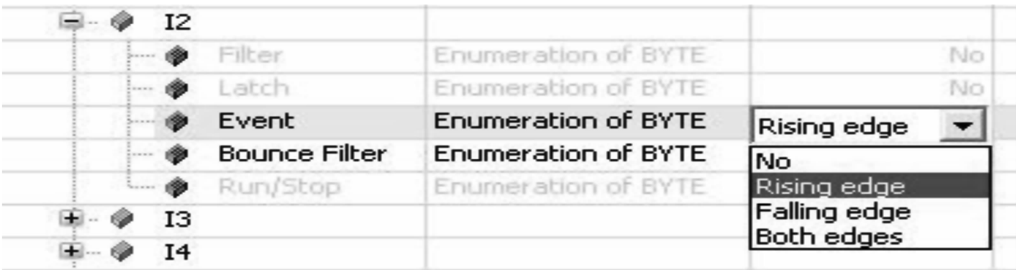


图 7-36　配置一个引起中断的端口

然后，在任务组态栏目建立一个事件任务 Event _ Task，如图 7-37 所示。

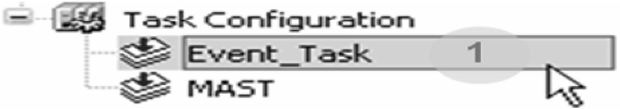


图 7-37　新建一个事件任务

双击事件任务“Event _ Task (1)”，打开任务配置，在任务类型单元，选择外部“External (2)”，然后在外部事件栏的下拉菜单中，选择“I2 (3)”，这就相当于把事件与外部端口联系起来。再把事件任务的程序 Event _ 1 加入到中断任务执行序列中，如图 7-38 所示。

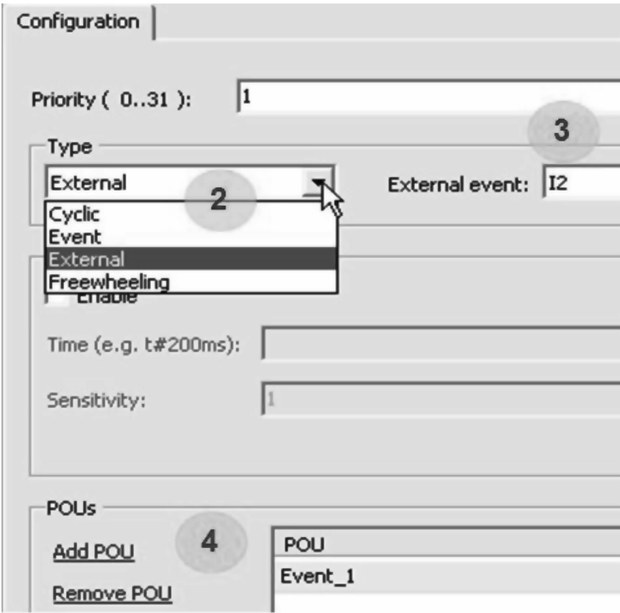


图 7-38　配置任务执行方式和优先级

在 M258 逻辑控制器和 LMC058 运动控制器上，我们可以在专家模板上对端口进行配置。

如在专家模板 DM72F0 上，点击鼠标右键打开下拉菜单，选择“添加设备”，并选择“Event _ Latch0”，建立一个子设备 Event _ Latch0。双击子设备（1），打开配置画面，在方式 Mode 上，选择事件“Event”，信号触发选择上升沿“Rising”（2），如图 7-39 所示。

事件任务配置（M258/LMC058）

在 M258 控制器和 LMC058 运动控制器上配置中断程序，与 M238 一样，首先把鼠标移到任务配置，然后点鼠标右键，打开菜单，选择“添加对象”，添加一个中断任务“Event _ Task”，如图 7-40 所示。

建立好程序名后，双击程序名打开程序配置，在程序类型栏目选择“External”（2），在外部事件信号 External event 栏目，选择“BLOCK0 _ I0”（3）。在 POU 程序组织单元栏目，选择把事件中断程序 Event _ 1 加入到此（4），如图 7-41 所示。

这样，我们就完成了一个中断程序的配置，当外部端口信号 I0 的状态发生变化，就会引起一个中断，使程序扫描进入中断程序，执行中断程序的指令。

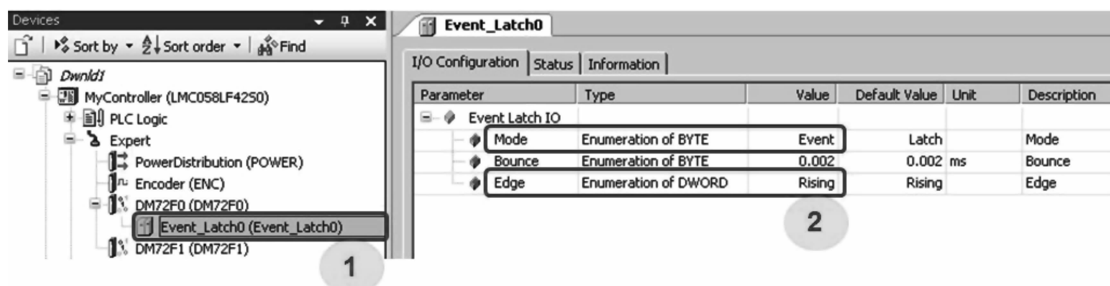


图 7-39　输入配置

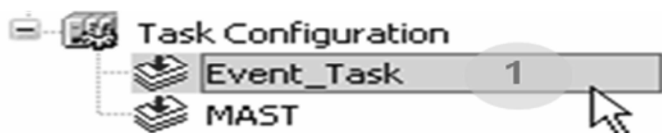


图 7-40　建立一个中断任务

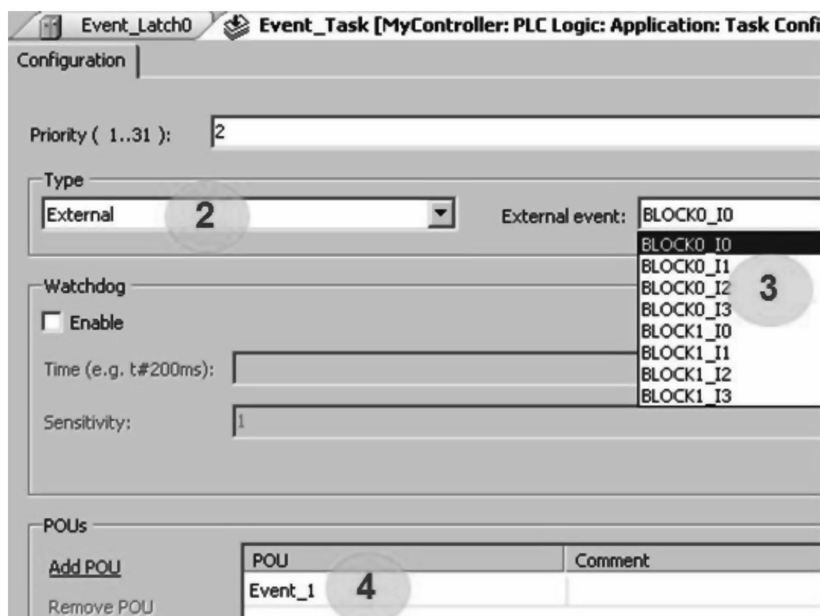


图 7-41　对任务进行配置

那么在中断程序里 I/O 是如何刷新的呢？我们再来看看引起中断的 I/O 配置。在设备 device 配置框，双击专家模板 DM72F0 打开模板配置并点击专家 I/O 映射 “Expert I/O Mapping”。在总线循环选择栏，选择扫描中断任务 “Event _ Task”，如图 7-42 所示。

然后，在中断程序里，我们要使用 PLC 系统应用库里的立即 I/O 读写功能，如图 7-43 所示。

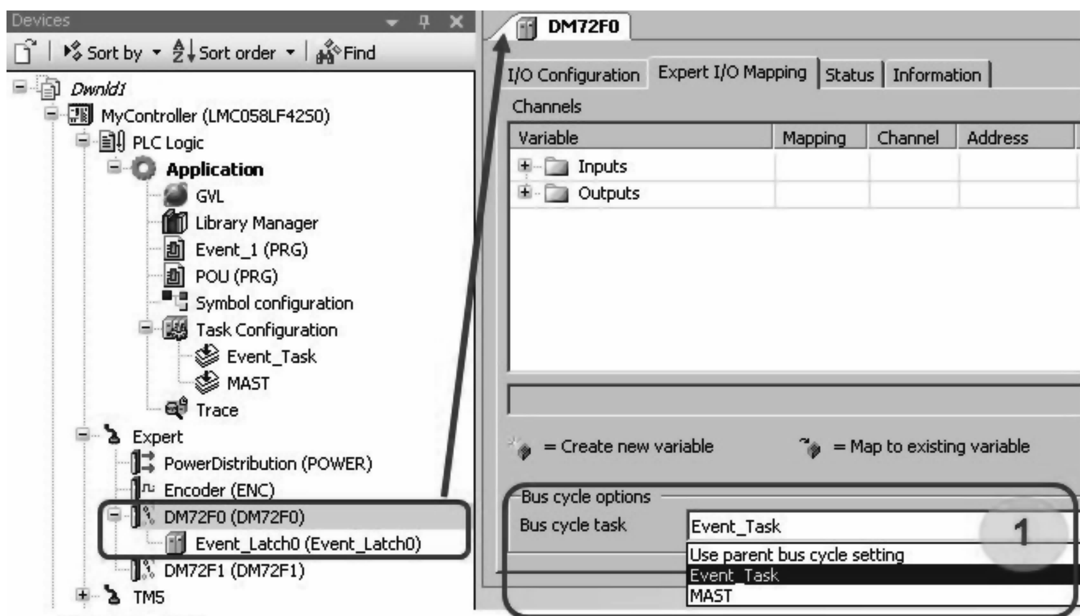


图 7-42　配置专家模板及总线扫描对象

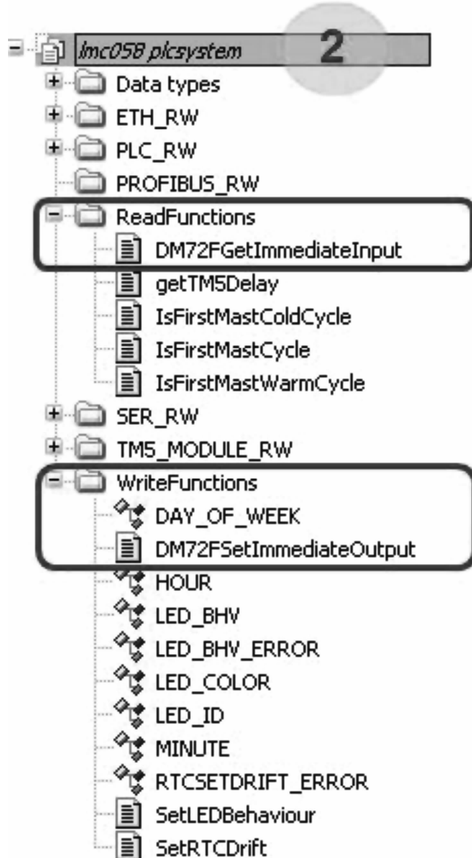


图 7-43　控制器系统应用库中的立即 I/O 读写功能

这些系统库中的快速 I/O 读写，不依赖总线扫描，只反映当前状态。

该功能适用于嵌入式专用 I/O 功能块 DM72F0 和 DM72F1。它将返回输入的当前物理值，当前物理值可能与该输入的当前逻辑值不同。该输入的变量值在下一个总线循环前不会改变。

立即输入功能如图 7-44 所示。



图 7-44　立即输入功能

功能块的输入见表 7-1。

表 7-1　功能块的输入

输　　入	类　　型	注　　释
功能块	INT	目标功能块 ● 0 = DM72F0 ● 1 = DM72F1
输入	INT	功能块的目标输入 0. . 6 = DI0. . DI6

功能块的输出见表 7-2。

表 7-2　功能块的输出

输　　出	类　　型	注　　释
DM72FGetImmediateInput	BOOL	功能块 <block> 的输入 <Input> 值 = FALSE/TRUE

功能块的输入/输出见表 7-3。

表 7-3　功能块的输入/输出

输入/输出	类　　型	注　　释
Error	BOOL	FALSE = 运行正常 TRUE = 检测到运行错误,功能返回无效值
ErrID	IMMEDIATE _ FUNC _ ERR _ TYPE	Error 为 TRUE 时检测到的运行错误代码

调用这些功能块如图 7-45 所示，在程序编辑区调入运算块，打开输入助手，点击模块调用，在 SEC 中选取立即读写功能块。

选取写功能块

此功能用于设置嵌入式专用 I/O（DM72F）的快速输出的当前物理值。

每个快速输出都有一个功能块：

DM72F0SetImmediateOutput0

DM72F0SetImmediateOutput1
DM72F1SetImmediateOutput0
DM72F1SetImmediateOutput1

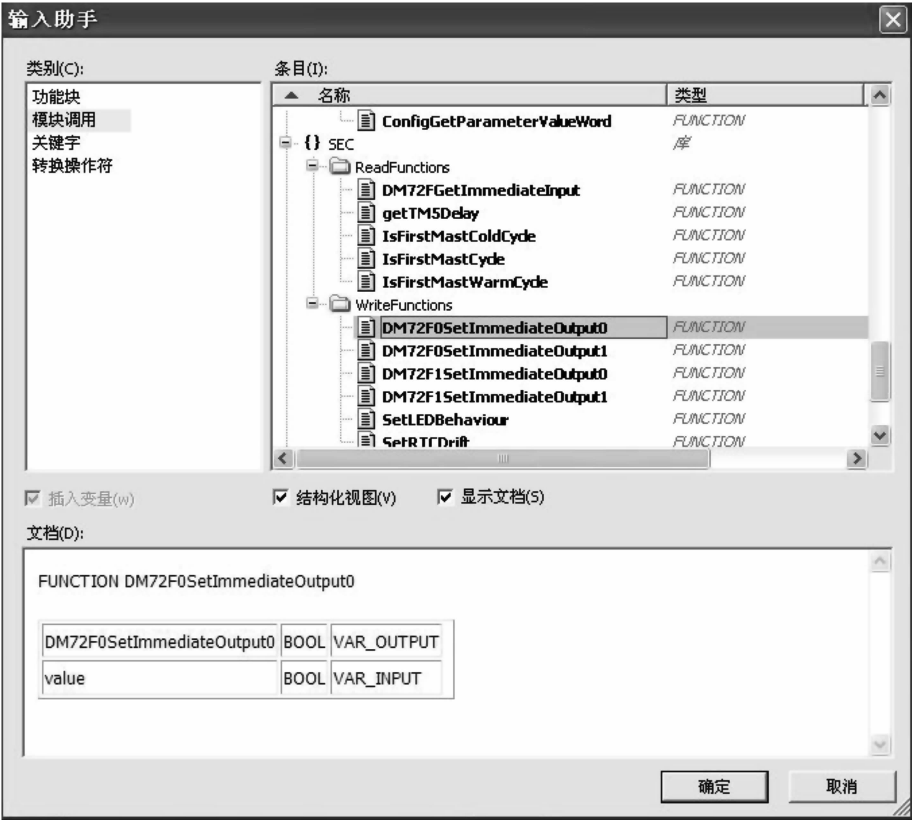


图 7-45　在模块调用 SEC 库中选取读写功能块

立即输出功能如图 7-46 所示。



图 7-46　立即输出功能

功能块的输入见表 7-4。

表 7-4　功能块的输入

输　入	类　型	注　释
值	BOOL	要求的输出值

功能块的输出见表 7-5。

实现 DM72F * SetImmediateOutput *

SoMachine 对于下列情况返回编译错误：

●在多个任务中使用 DM72F * SetImmediateOutput * 功能。

●在应用程序中使用与 DM72F * SetImmediateOutput * 关联的% Q。

●该输出已经专用于某一嵌入式专用 I/O 功能块功能（例如：PWM、频率发生器、编码器的反射输出、警报）。

表 7-5　功能块的输出

输　　出	类　　型	注　　释
DM727Fb′SetImmediateOutputn[2]	BOOL	TRUE = 设置物理输出值

（1）b 是目标功能块：

● 0 = DM72F0

● 1 = DM72F1

（2）n 是功能块的目标输出：

● 0 = DO0

● 1 = DO1

第 8 章　程序的调试和诊断

程序编辑完成后，在实际运行程序时，有时会出现各种状况。比如，应该输出时，并没有输出或没有及时输出。程序运行一段时间会死机或报警，这都是如何引起的，它的来龙去脉又是怎样的？我们可以通过下面介绍的方法来诊断和调试程序，使之完全按照设计的构想来运行。通过这些方法，达到我们对程序运行的可控和可观。

8.1　状态栏

在线登录到控制器上或仿真登录控制器后，在程序运行区下部会看到控制器状态和程序运行状态。如图 8-1 所示，我们仿真登录到控制器上，可以看到控制器处于仿真状态，程序处于停止。

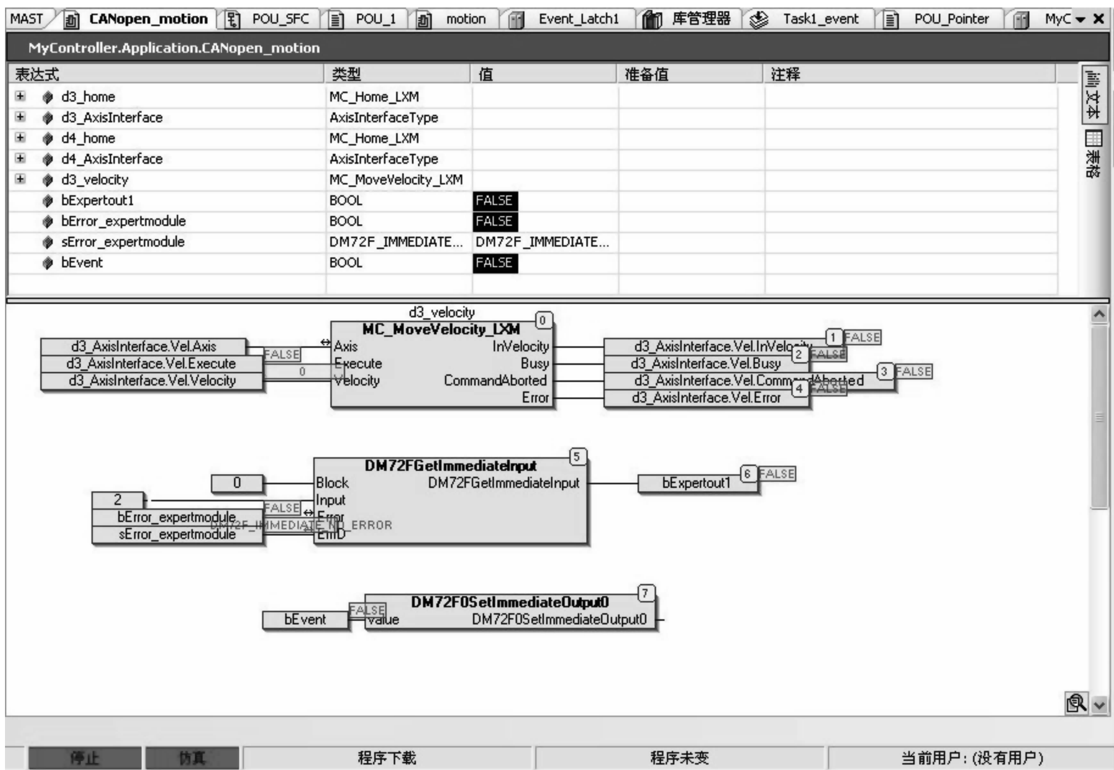


图 8-1　控制器状态和程序停止状态

在图 8-2 中可以看到程序运行时，下部显示的状态。

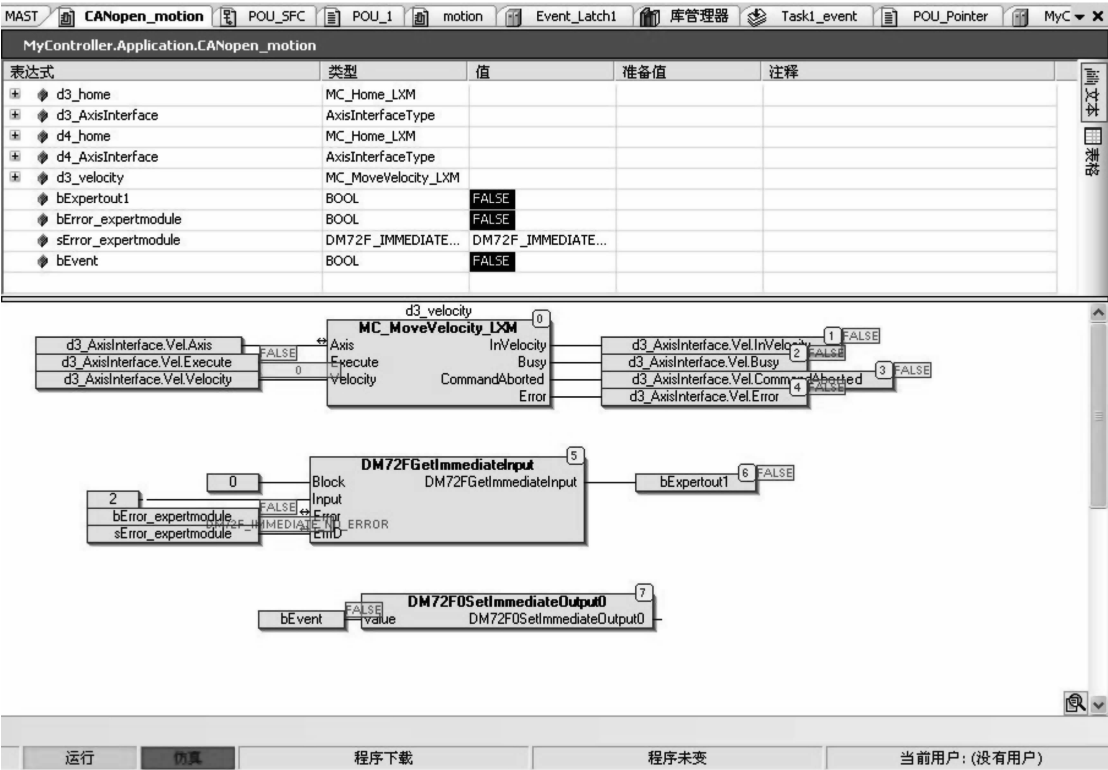


图 8-2　控制器在仿真状态并且程序运行

8.2　信息窗

在程序栏目下，打开视图菜单，选择“消息”，如图 8-3 所示。

在打开的消息窗，你可以看到程序编译后有关错误、警告和信息的报告，还可以看到预编译的提示，如图 8-4 所示。

当编辑的程序有问题时，在消息栏会给出提示，双击这个提示会找到出错的地方，如图 8-5 所示。

用户还可以应用编译指示 pragma 命令来影响编译代码的生成。例如，你希望一个变量的数值显示可以是 16 进制或是 10 进制。有关这方面的知识可以查找程序的在线帮助。

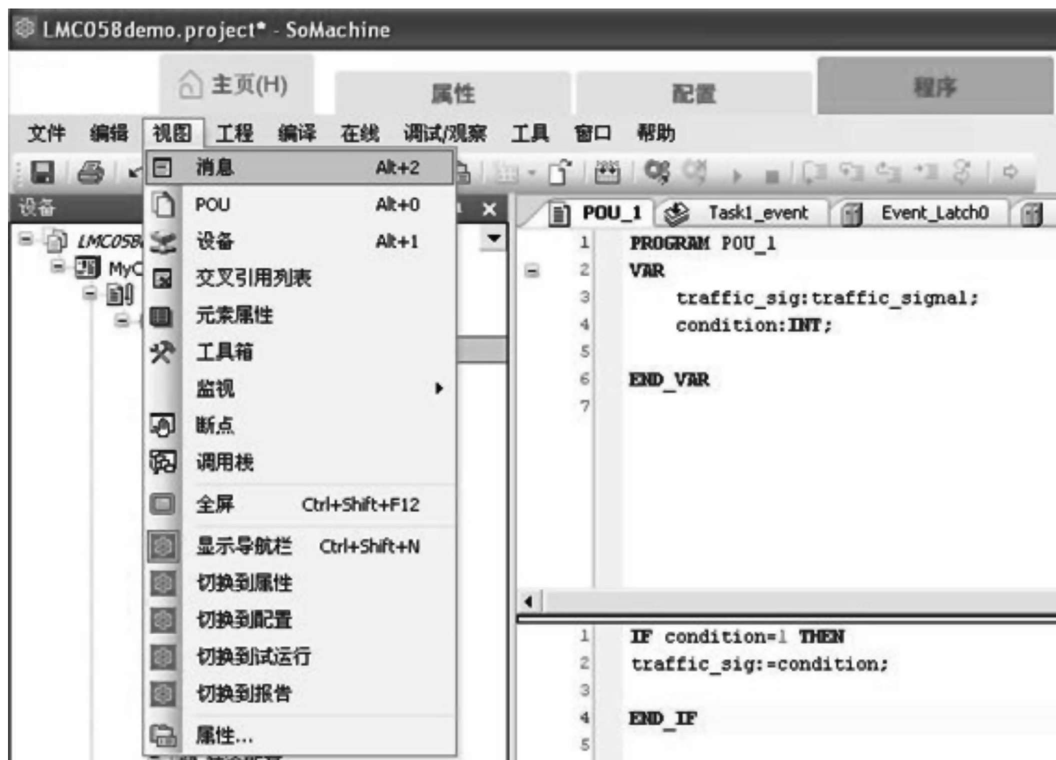


图 8-3　选择打开消息窗

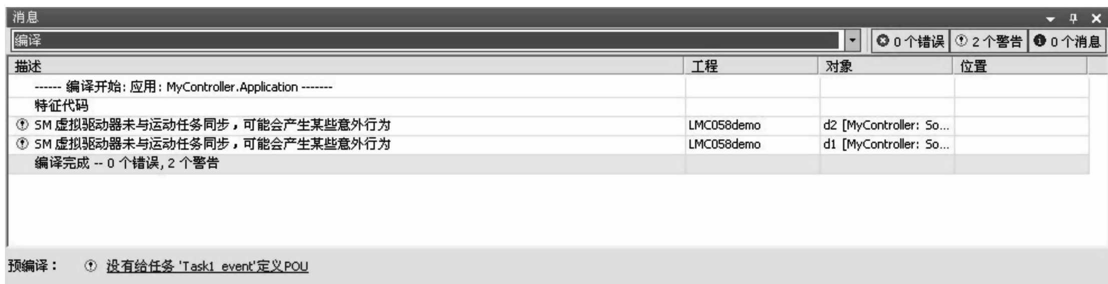


图 8-4　信息报告

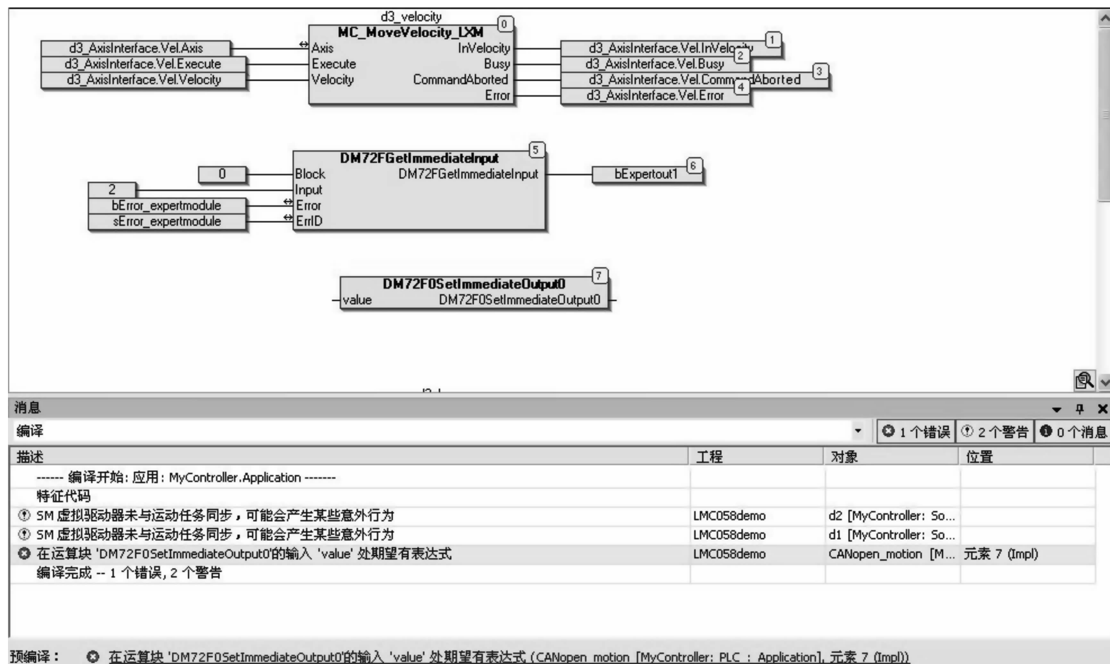


图 8-5　通过双击错误提示找到错误源

8.3　日志

在线登录到控制器（M258 逻辑控制器/LMC058 运动控制器，IMC）上，在控制器管理栏目（MyController）上点击日志（Log）你可以看到如下信息。

系统事件：错误提示，警告，例外，信息。

I/O 驱动的输入：一些支持运行的组件。

客户指定输入：一些支持应用的组件。

如图 8-6 所示选择组件的分类，可以过滤不同组件的显示。也可以在线刷新日志和把日志导入和导出。

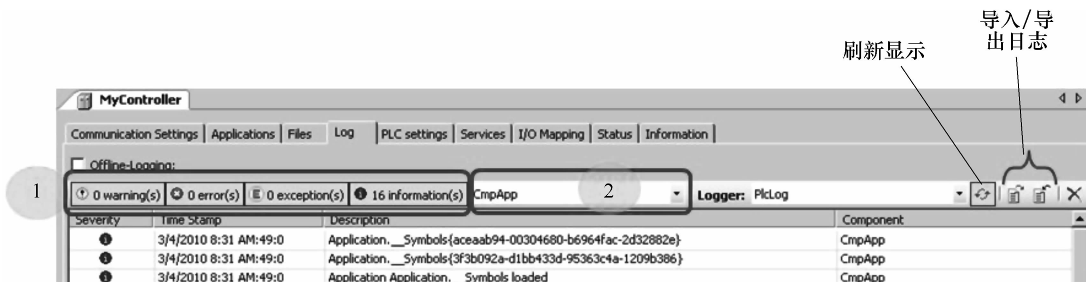


图 8-6　日志的显示

8.4　调试工具

在一个程序编制完成后，就要在线运行，以检验程序的执行是否按照设计指标安全运行。但有时会由于某些外部条件还没有准备好，不足以实现整个程序的运行。这就要求我们可以脱离外部条件进行控制器的仿真运行。或对变量进行强制执行并进行检测来查看运行状态和效果。有时，由于程序很大，我们可以设置程序断点来分部调节。在观察程序运行时，我们还可以对感兴趣的变量或命令进行曲线纪录，类似于把执行过程拍摄下来，以便分析。这就是我们在程序调试时用到的工具。这些工具非常重要，它是我们实验程序，检验结果或测试功能的有力手段，熟练地使用这些工具，往往会达到事半功倍的效果。

8.4.1　仿真

对于程序的仿真，我们应该是很熟悉了。在前面的章节中，我们使用的案例大量地用到了仿真，这使你可以很方便地模仿和验证程序运行。它的用法就是在程序栏目下，打开在线菜单，选择仿真，如图 8-7 所示。

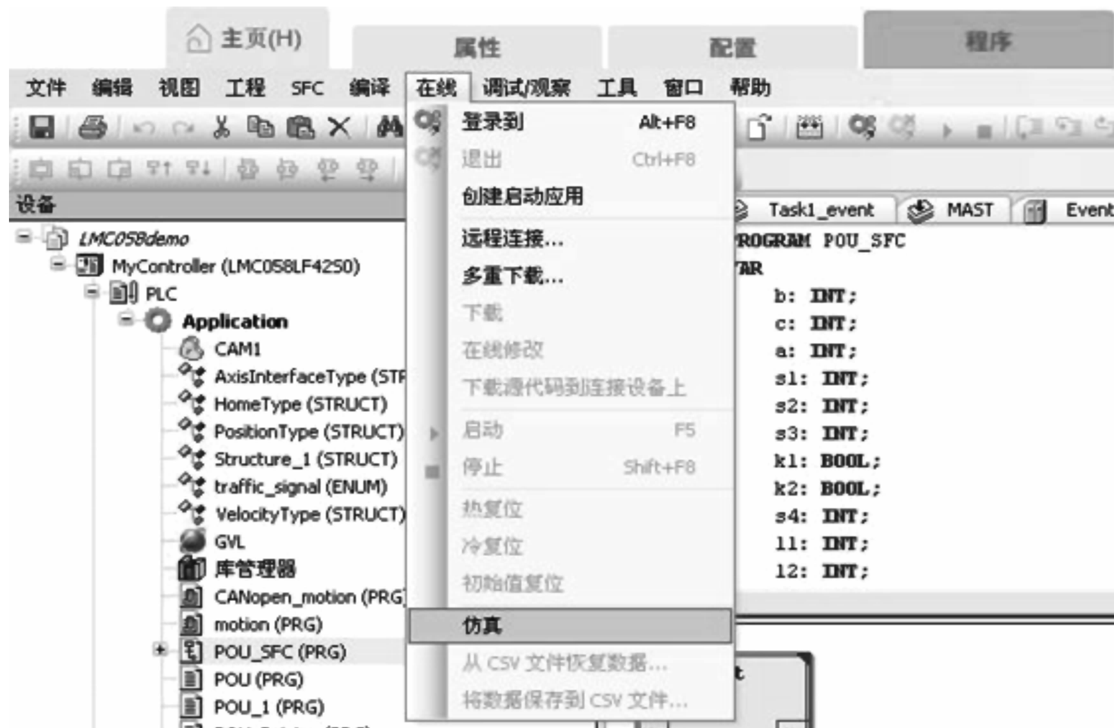


图 8-7　选择程序仿真

进入程序仿真后，在程序下部控制器状态会显示为“仿真”，然后再打开在线菜单，选择登录并启动程序就可以仿真程序运行，验证程序执行和变量的变化。

8.4.2　变量监测和强制

在观察变量的变化或强制状态时，我们可以配置一些观察表，把我们需要在同一情况下要观察或强制的变量集合起来，一起观察并比较。如图 8-8 所示，在程序栏目，打开视图菜单，点击“监视”后选择一个监视表，例如“监视 1”。



图 8-8　选择监视列表

在打开的监视列表中，点击表达式下的空栏，会出现输入助手。通过输入助手选择要监视的变量，组成列表。在线登录控制器后，就可对变量进行强制，如图 8-9 所示。

监视 1

表达式	类型	值	准备值	注释
MyController.Application.POU_SFC.a	INT	F 10		
MyController.Application.POU_SFC.b	INT	F 20		
MyController.Application.POU_SFC.c	INT	0		
MyController.Application.POU_SFC.k1	BOOL	F TRUE		

图 8-9　变量列表及在线强制

强制值写到准备值栏目中，然后通过以下快捷键完成写入。

Ctrl + F7：写入值；

F7：　　　强制值；

Alt + F7：释放值。

8.4.3 设置断点（Breakpoint）

在程序执行时，为了分部检查程序的执行结果，我们可以在程序中设置断点。即当程序执行到断点时，执行暂停，并在程序下部状态区显示断点暂停（HALTONBP），当要继续时，再启动程序运行，程序即从断点处开始继续运行程序或到下一个断点处暂停。断点状态指示如图 8-10 所示。



图 8-10 断点状态指示

我们选择设置断点如图 8-11 所示，即在程序功能下，打开视图菜单，选择断点。



图 8-11 程序建立断点

点击断点后，即打开断点设置框，如图 8-12 所示，在这个框内可以进行断点设置。



图 8-12 断点配置框

点击框中的新建，开启如图 8-13 所示设置框。

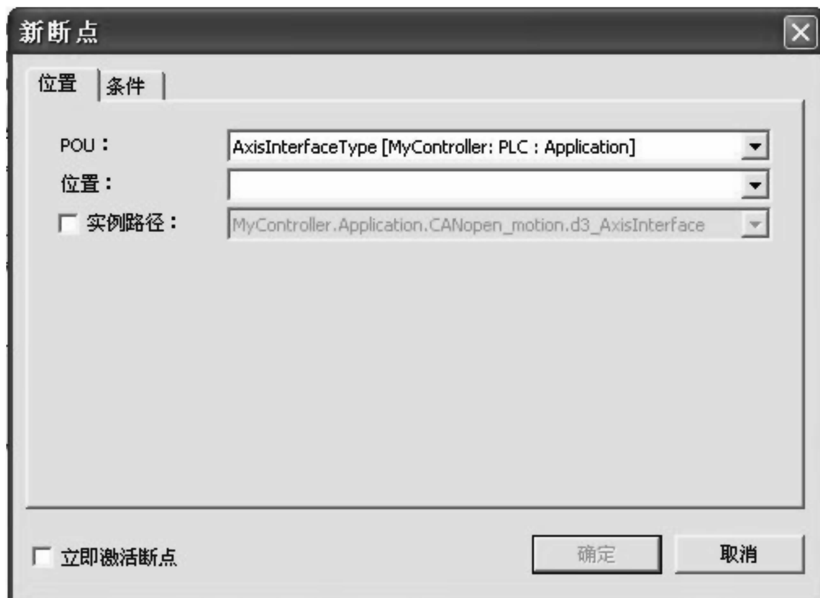


图 8-13　新断点设置框

在此设置框中点击位置，并在 POU 菜单选出需要设置断点的程序，在位置菜单选择设置断点位置，如图 8-14 所示。



图 8-14　断点程序，位置设置

确认设置后，断点配置框如图 8-15 所示。

仿真并运行带断点的程序，这时的断点配置框如图 8-16 所示，可以看到断点还没有被激活。

点击配置框中的程序名，这时配置框中的图标被激活，如图 8-17 所示。

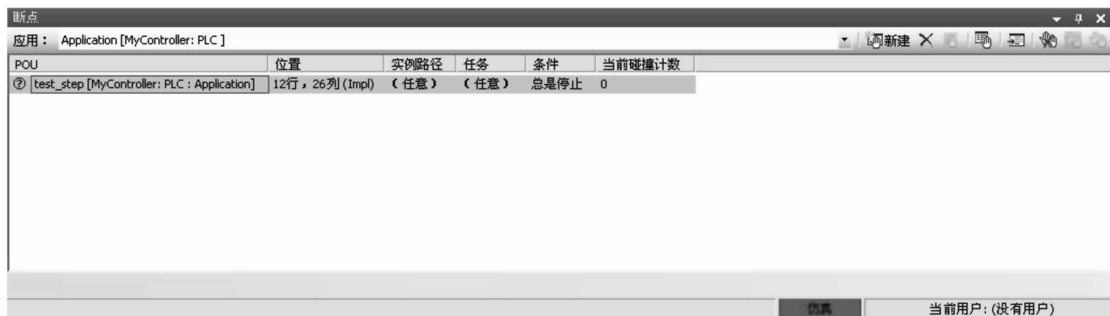


图 8-15　断点配置完并启动仿真

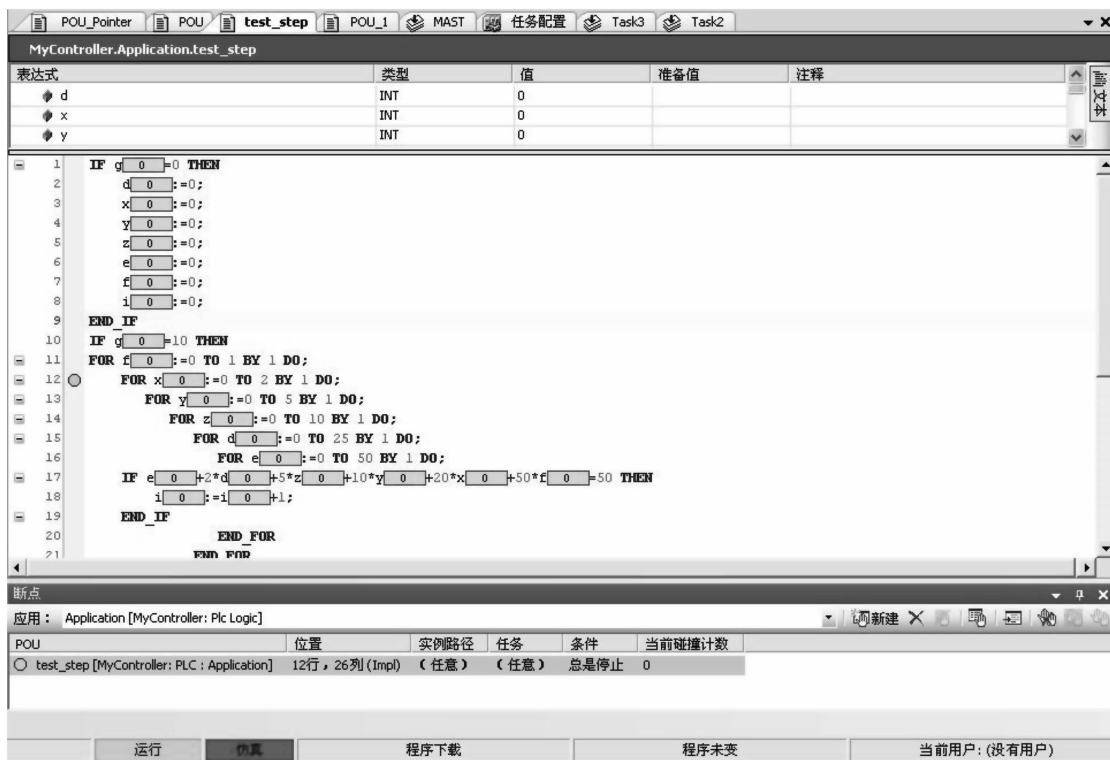


图 8-16　断点配置框状态



图 8-17　激活配置图标

点击配置框中激活断点的图标，这样程序的断点才能有效，如图 8-18 所示。

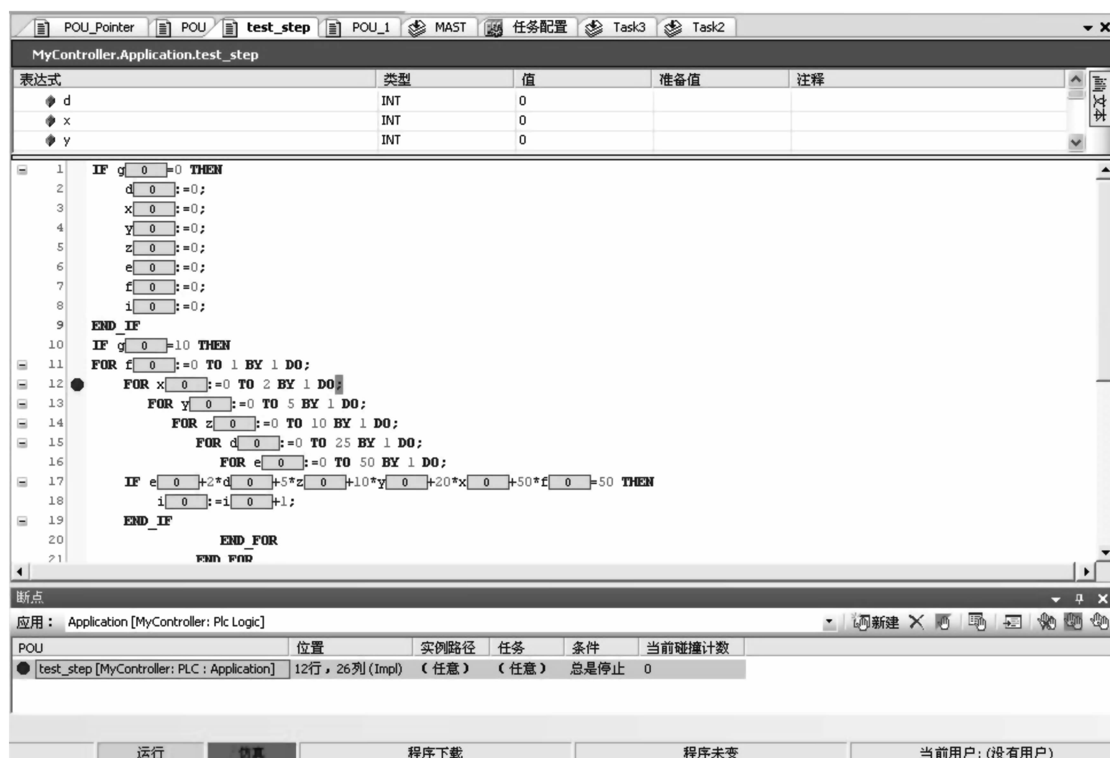


图 8-18　激活程序断点

给程序条件启动运行后，程序运行到第一断点后停止，如图 8-19 所示。再次启动程序运行，程序到达第二断点后停止，如图 8-20 所示。再次启动，程序到达第三断点停止，如图 8-21 所示。再次启动，程序又到达第四断点停止，如图 8-22 所示。依次下去，我们就会看到程序的一步步执行情况了。

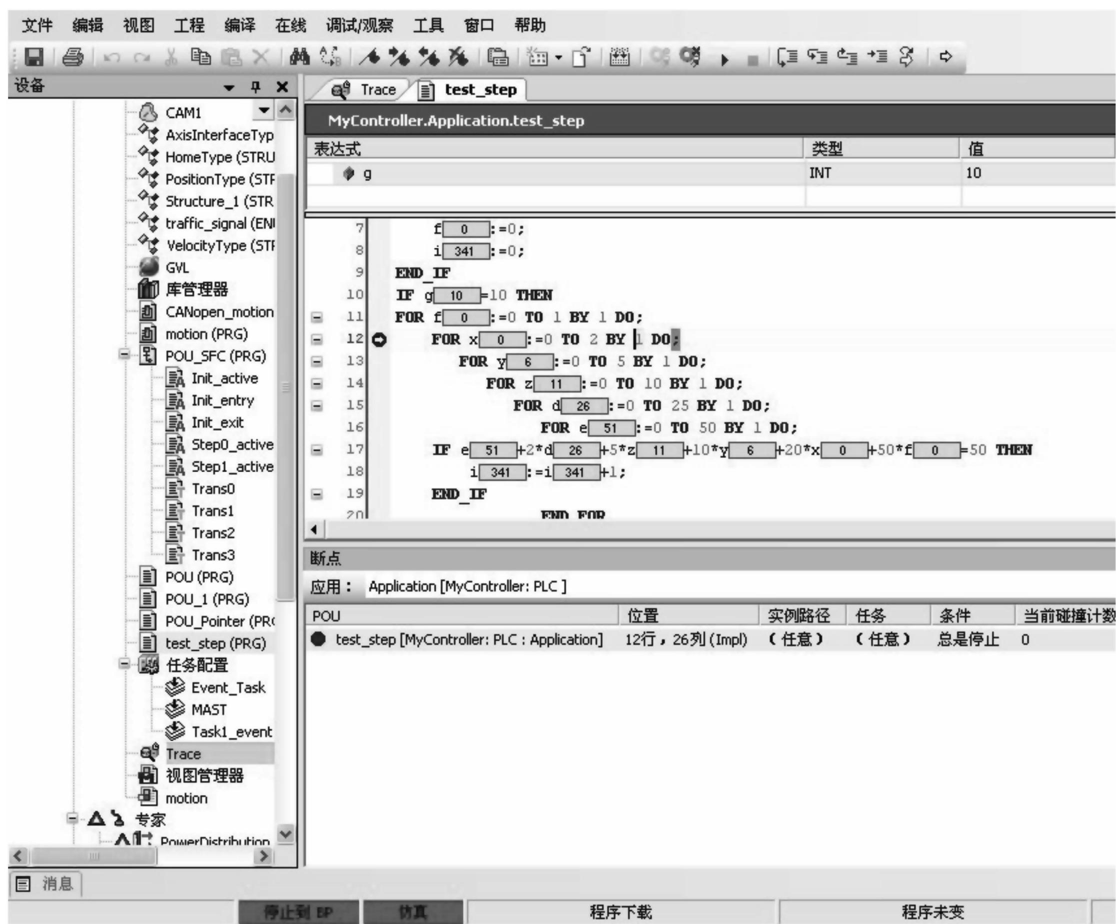


图 8-19　程序运行到第一断点停止

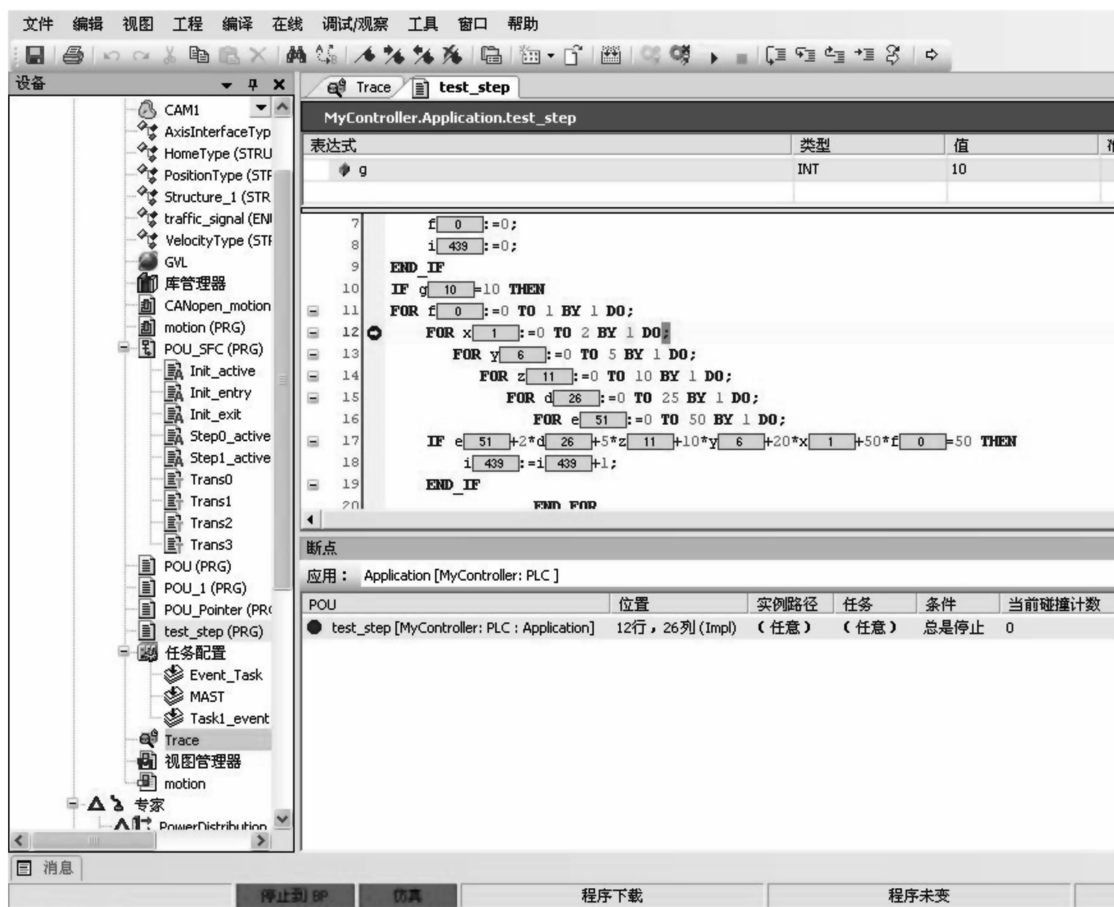


图 8-20　程序运行到第二断点停止

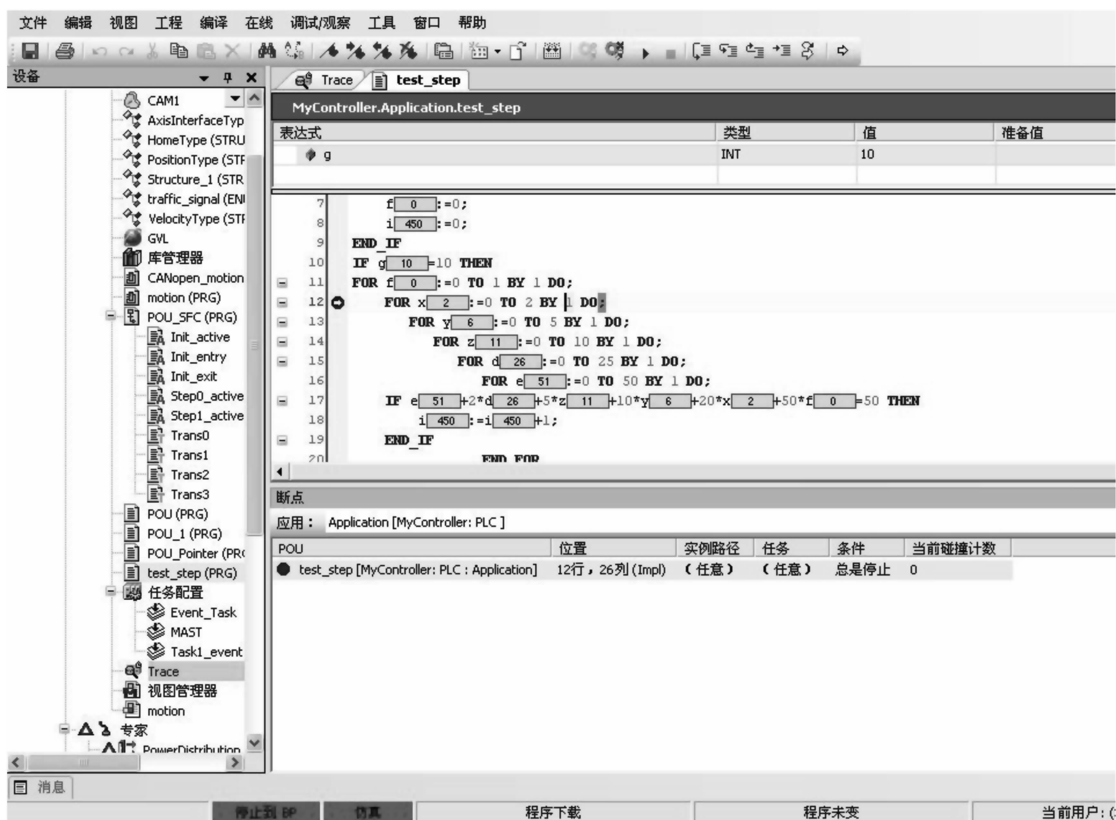


图 8-21　程序运行到第三断点停止



图 8-22　程序运行到第四断点停止

我们再来看一个设置断点的案例。首先我们打开断点设置，如图 8-23 所示。



图 8-23　设置一个新断点

然后，在打开的断点设置栏目选择要加载断点的程序和确定断点位置，如图 8-24 所示。



图 8-24　配置断点

确定断点配置后，启动程序仿真如图 8-25 所示。

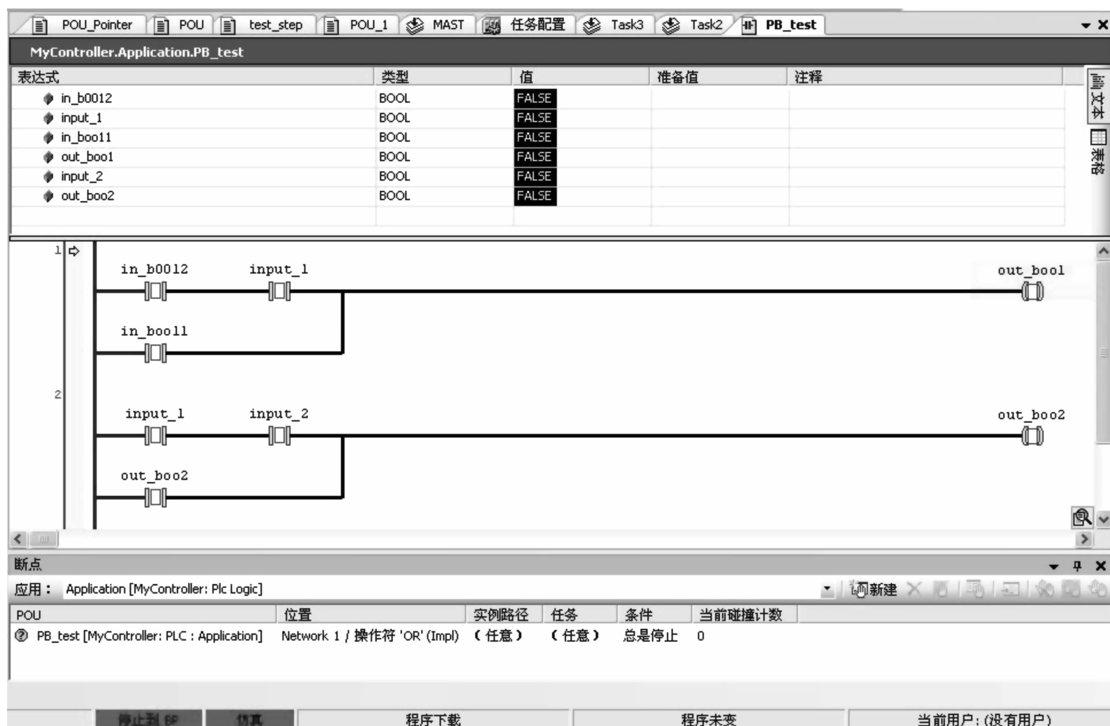


图 8-25　启动程序仿真

选择第一段程序为断点位置，即点击一下程序的第一段，如图 8-26 所示。

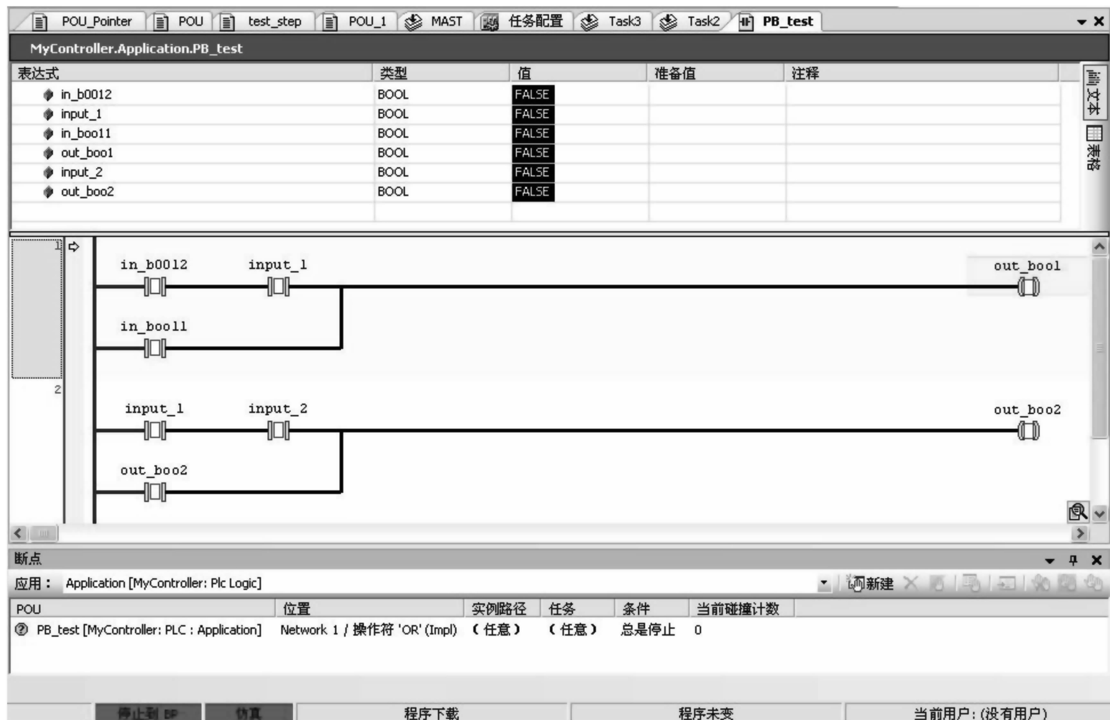


图 8-26　选择执行断点的程序段

切换断点操作，如图 8-27 所示。

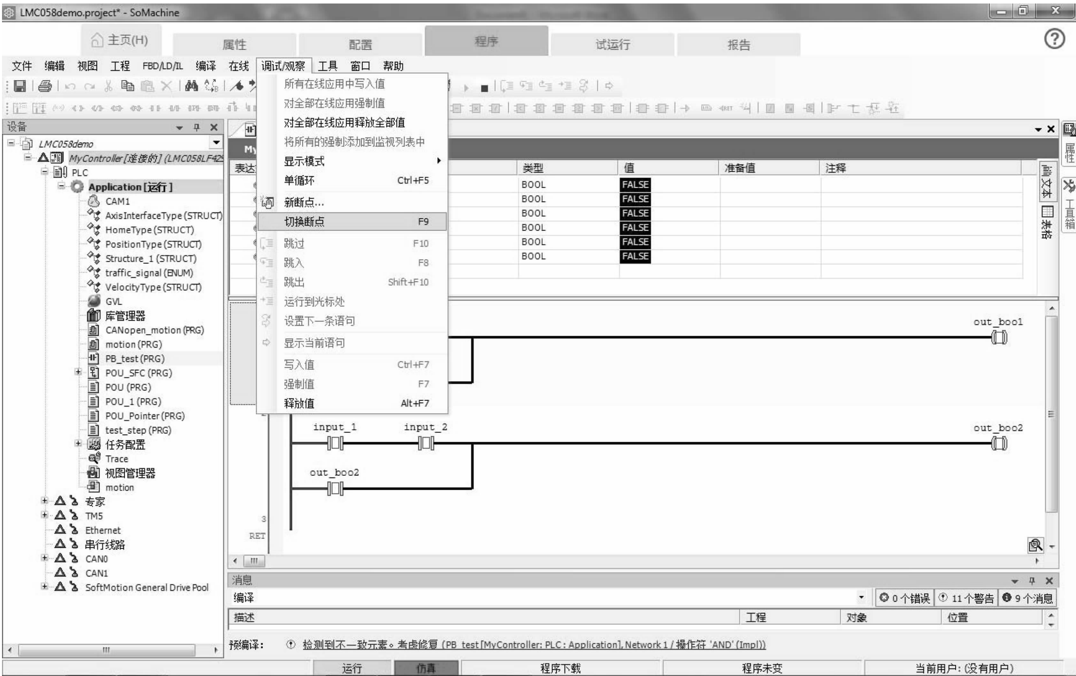


图 8-27　切换断点操作

通过切换断点操作，使得调试程序段的断点被激活，如图 8-28 所示。

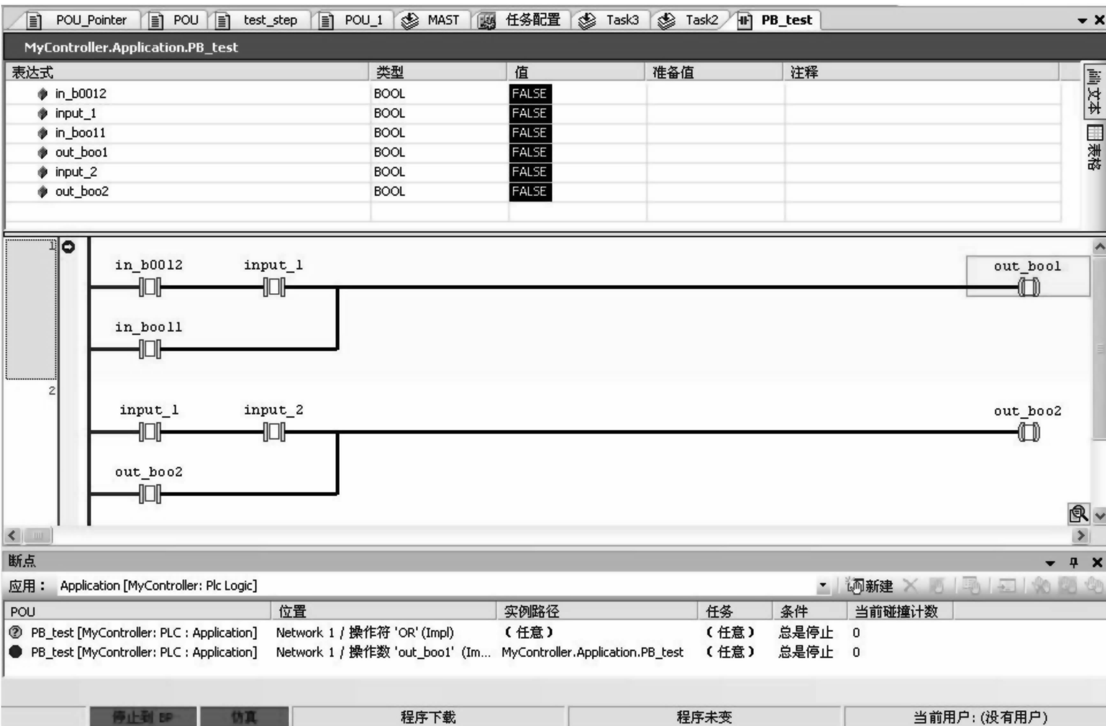


图 8-28　激活断点

选择“单循环”，如图 8-29 所示，这样程序进行单步运行。即按一次运行，程序执行一个周期即停止并等待下一次运行指令。

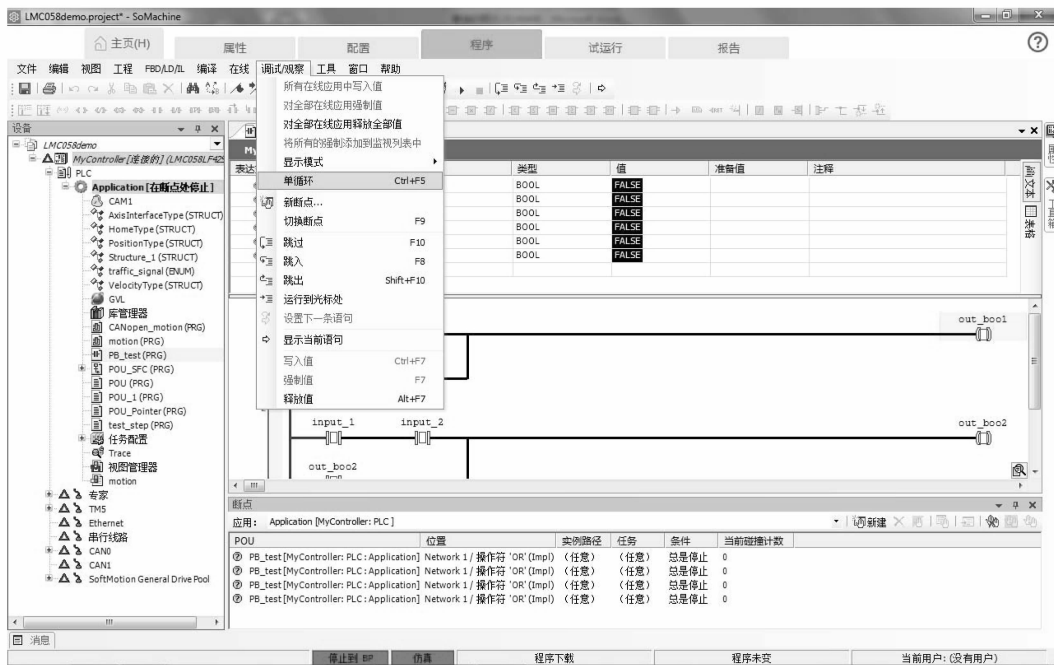


图 8-29　选择单循环

点击“单循环”后，弹出确认窗口如图 8-30 所示。

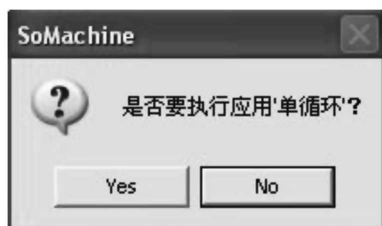


图 8-30　确认窗口

确认后，程序执行一步，如图 8-31 所示。

加入程序中一个条件，再次点击控制器启动，弹出确认窗口，如图 8-32 所示。

确认后程序执行一步，如图 8-33 所示。

再加入程序中一个条件，再次点击控制器启动，弹出确认窗口，如图 8-34 所示。

确认后程序执行一步，如图 8-35 所示。

通过不断启动程序，使之一步一步地执行以查看程序是否按照设计进行工作。

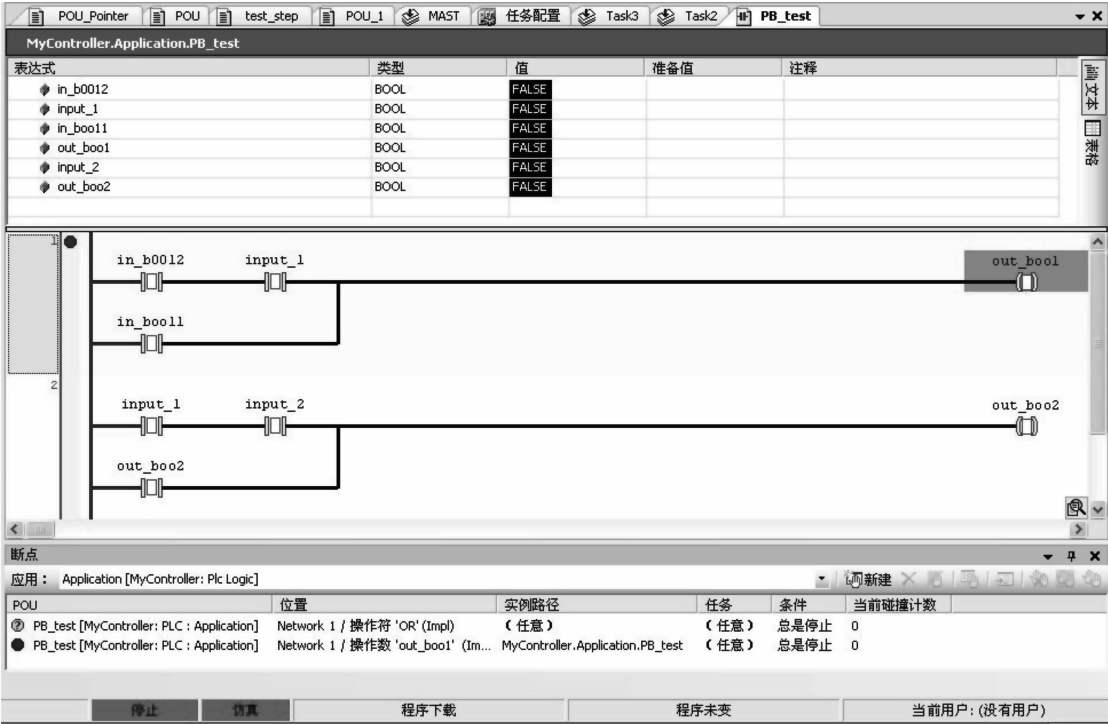


图 8-31　程序执行一步后停止

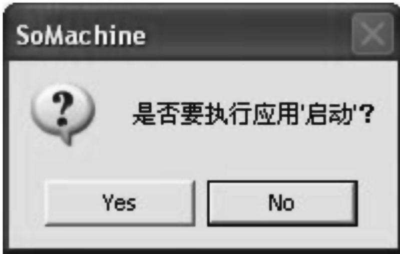


图 8-32　确认窗口

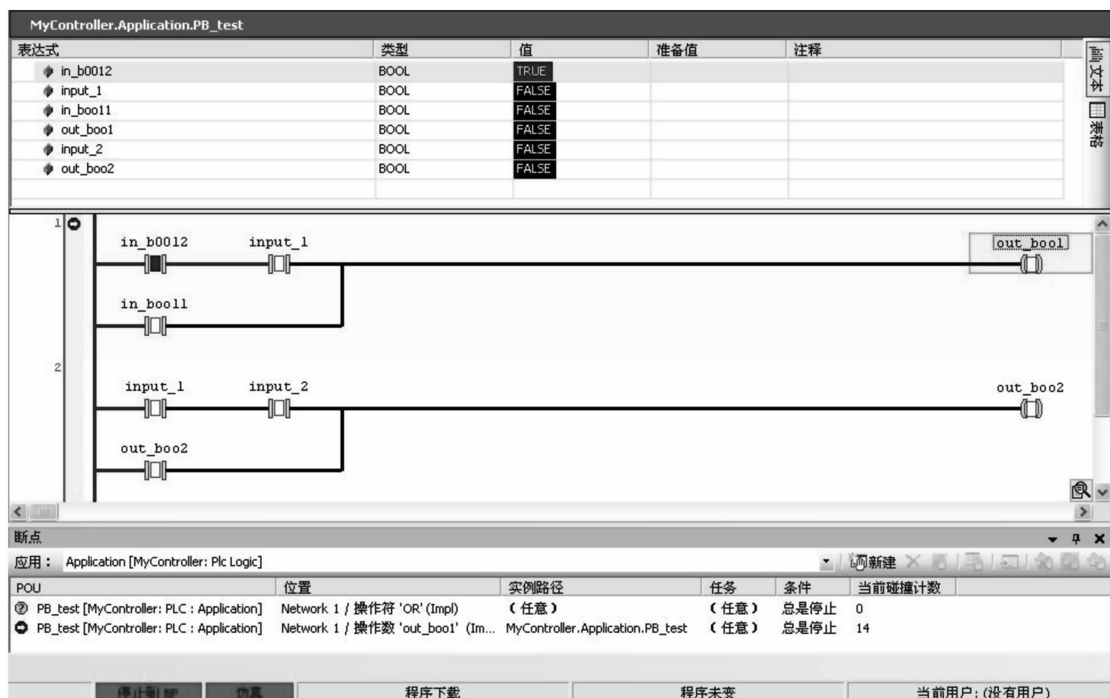


图 8-33　程序运行一步后停止

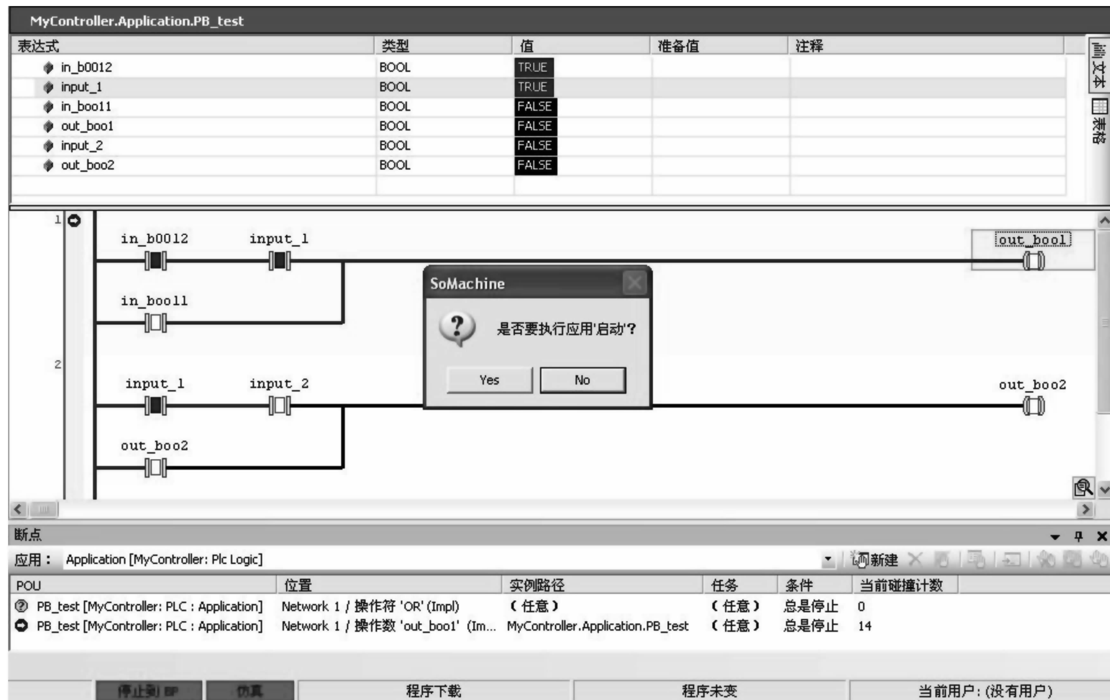


图 8-34　确认窗口

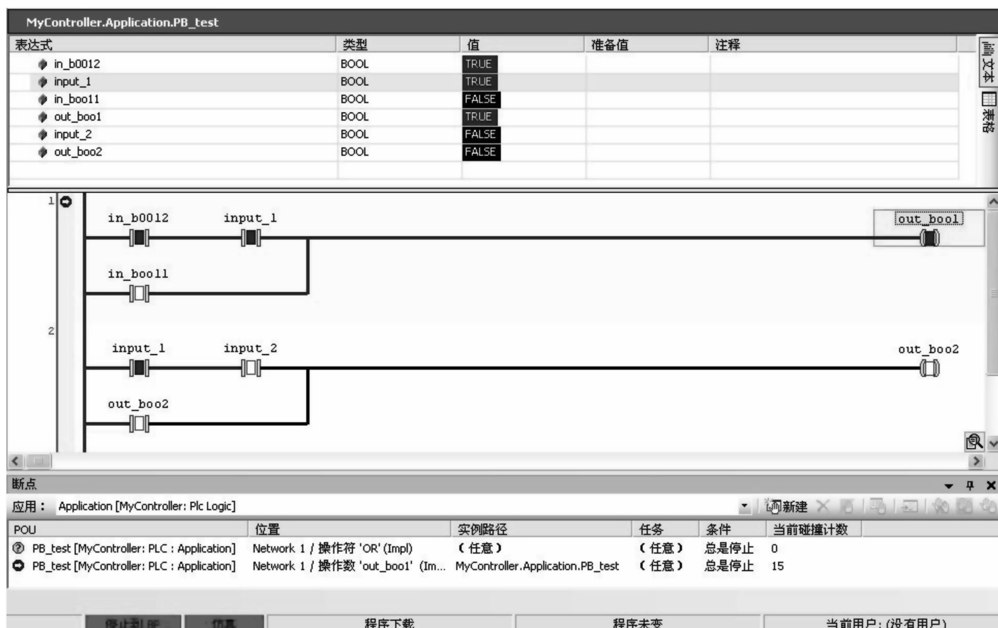


图 8-35　程序再次执行一步

8.4.4　曲线记录

在程序的调试和诊断过程中，还有一个工具是非常实用和有效的，即曲线记录。它相当于把一个程序的执行过程拍摄下来，其中的演员就是那些你要关注的命令字、状态字、电动机运动的速度、位置、各种传感器信号等。通过对这些状态量、过程量的拍摄记录，可以非常清晰地看到程序在整个执行过程中，这些变量在各个时段的情况，如图 8-36 所示。

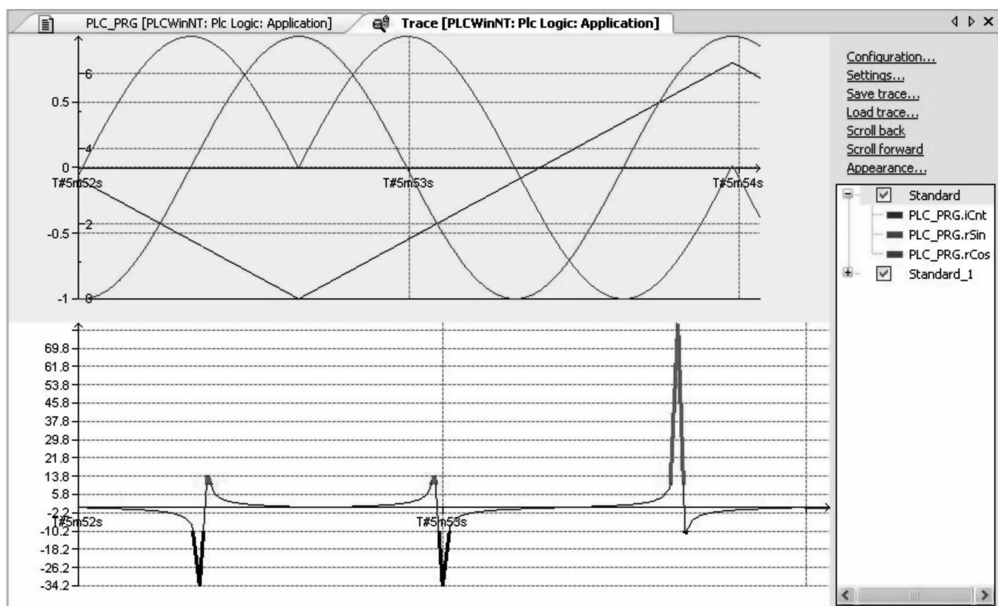


图 8-36　程序执行过程的曲线记录

建立一个曲线记录

1）鼠标放置在应用（Application）点右键打开下拉菜单，在菜单中选中添加对象，并按导引出的菜单找到跟踪并按鼠标左键确认，如图 8-37 所示。

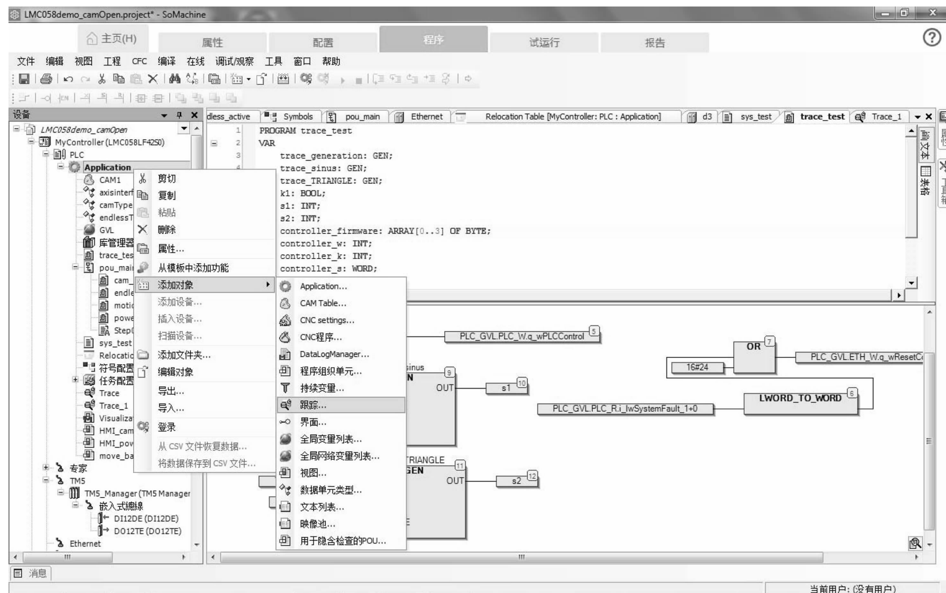


图 8-37　建立曲线跟踪

2）确认跟踪后，会弹出如下对话框，在此框中，填写跟踪的名称，比如 Trace _ 1，如图 8-38 所示。



图 8-38　命名跟踪名称

确认名称并打开，弹出如下配置框，如图 8-39 所示。

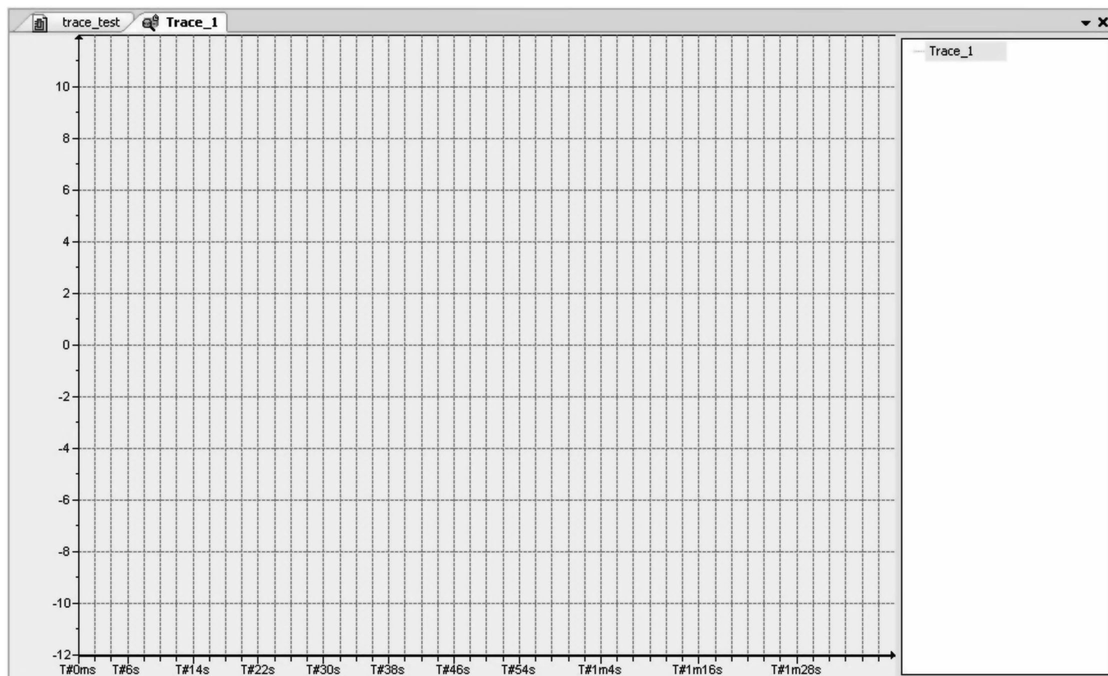


图 8-39　新建的跟踪配置框

3）配置跟踪设置。在跟踪名处，点击鼠标右键打开菜单，如图 8-40 所示。

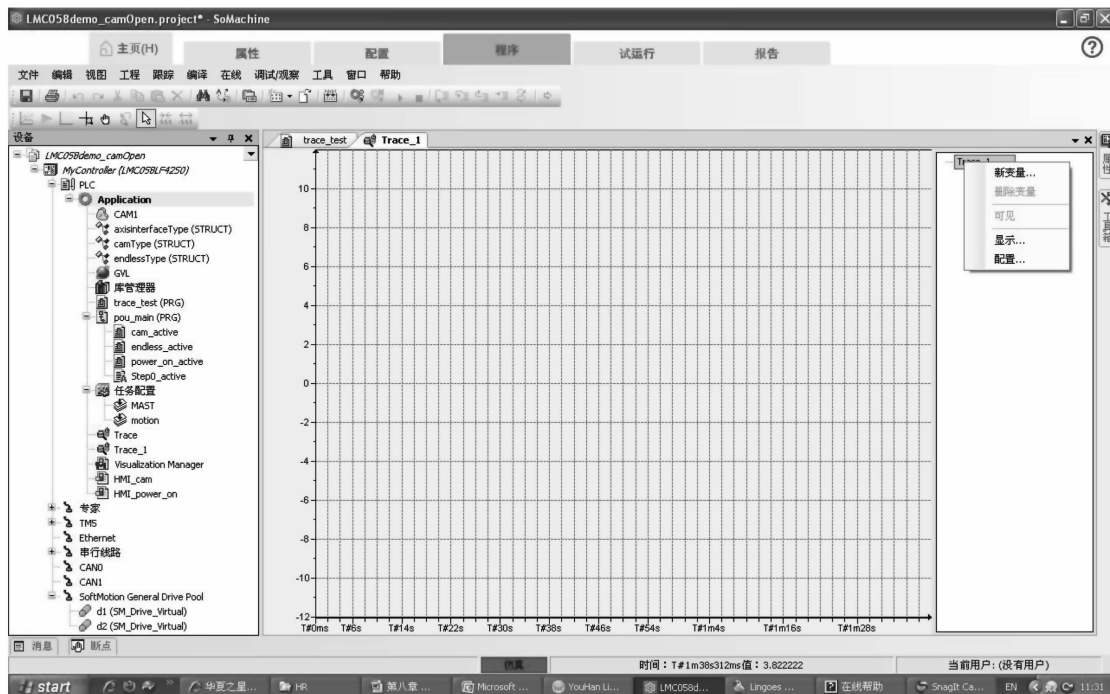


图 8-40　打开一个跟踪的配置

4）在图 8-41 中，点击配置，打开配置框。在此配置框中有触发变量的选择，即记录的启动是基于这个变量状态的变化的。比如，记录一台电动机起动过程中，速度、位置的情况。我们就可以定义电动机的速度为触发变量，触发沿为速度的上升沿，触发器水平为 10，即当电动机速度上升到 10 后即启动电动机速度、位置记录。当然，我们也可以定义启动电动机的信号作为触发信号，触发沿为上升沿，触发水平为启动信号接通，即“1”或 true。在跟踪配置中，触发位置是指触发后的曲线记录占整体记录画面的百分比。比如说 50%，即在整个记录画面，启动记录是从记录画面的中间开始的。

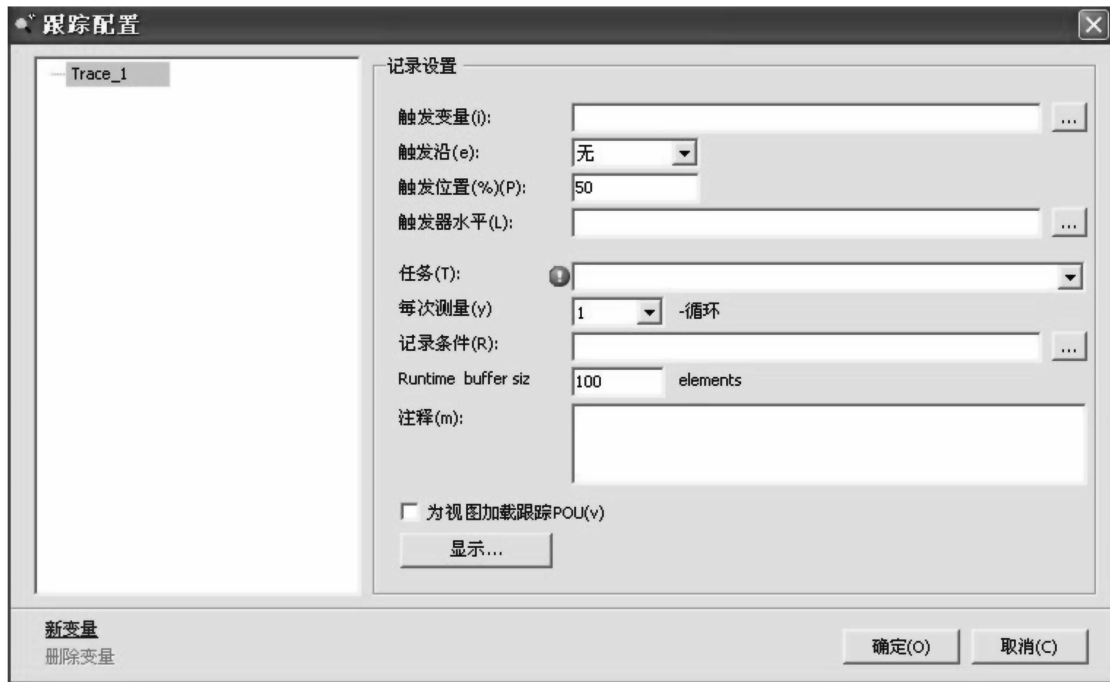


图 8-41　跟踪配置框

5）在图 8-42 所示的配置框中填写触发变量，可以点击右边的图标，打开输入助手。

借助输入助手，可以选择将要观察的变量，比如 k1。

6）根据此配置框，来配置参数。跟踪配置如图 8-43 所示。

各栏目定义说明如下：

触发变量：配置触发变量是可选的，它与其他一些条件共同决定了跟踪的时间范围。

下面进行详细解释：

该变量可以是一个布尔变量、一个表达式或一个模拟变量，也可以输入枚举变量或属性变量。当该变量满足了定义的值——该值根据“触发边沿”（定义见下文）类型来决定，跟踪将在采样一段时间后停止——该采样时间段由“位置”（定义见下文）的百分比来决定。也就是说，一旦触发变量变为真或满足某一特定值，跟踪将继续一段定义好的周期。

点击按钮[...]使用输入助手来选择一个合适的触发变量。

注意，通过“记录条件”（定义见下文）也可以控制跟踪的开始。

触发边沿：



图 8-42　配置触发变量

无：无触发（默认值）。

上升沿触发：　布尔型触发变量的上升沿，或模拟触发变量增大到为它所定义的“水平”值，触发事件发生。

下降沿触发：　布尔型触发变量的下降沿，或模拟触发变量减小到为它所定义的“水平”值，触发事件发生。

上升/下降沿触发：　在上升沿触发或下降沿触发中描述的条件达到时（如上所述），触发事件发生。

触发位置（%）：触发事件发生后，要记录跟踪变量的测量值百分比。即定义，触发事件发生前所要记录的跟踪变量测量值的百分比，以及触发事件发生后要记录的百分比。例如：填入 25，则在跟踪曲线中，75% 是触发事件发生前的测量值，25% 是触发事件发生后的测量值。如果希望一发生触发事件，就开始跟踪，那么填入 100。

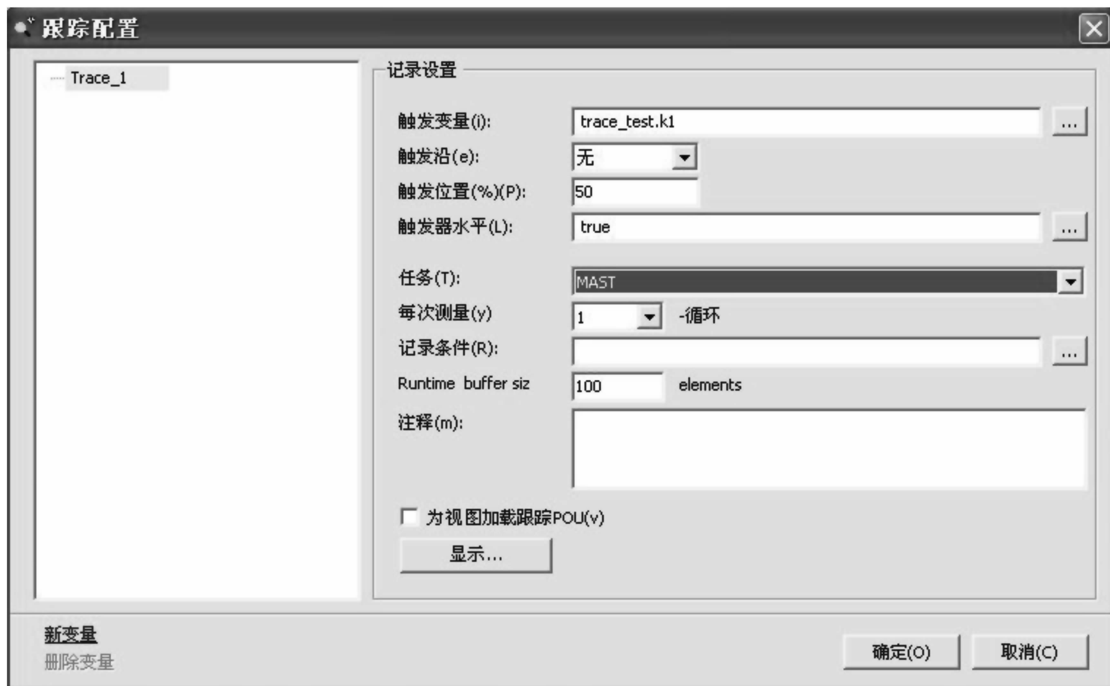


图 8-43　跟踪配置

触发水平：若使用模拟量作为触发变量，在此处定义该变量为多少时产生触发事件。可直接输入一个数值，或用变量来定义该数值（允许使用枚举型常量）。默认值：空。

任务：在可用的任务列表中选择，该任务被执行后从中读取出跟踪变量的值。默认值：任务配置树中的第一个任务。

每次测量：此处定义再次读取变量前我们需要等待的周期数，通过（缓冲区大小 * 第 x 个周期处测量 * 任务间隔）计算公式，可以计算出最小的监视时间段。

记录条件：此处可输入一个布尔变量、一个数值或一个布尔表达式。该条件为真，则启动跟踪采样。若此处没有任何输入，则在下装跟踪配置并且应用开始运行后，立即开始跟踪记录。

缓冲区大小：字节数，运行系统中分配给当前记录用以保存跟踪变量值的缓冲区大小。监视跟踪时会显示此缓冲区中的内容。默认值：100。

注释：在此输入有关当前记录的注释文本。

显示：点击此按钮将打开“编辑显示 <记录名>”对话框，在该对话框中可以为当前配置的记录设置其跟踪窗口的显示（坐标轴，颜色，滚动动作），如图 8-44 所示。

为视图加载跟踪 POU：勾选其复选框，引发对组件 <跟踪名>_<任务名>_VISU 的隐式编译。如果想在视图中集成跟踪的输出，可使用该选项。

7）配置要记录的变量，如图 8-45 所示。

如图 8-45 所示，把鼠标移到记录名上点击右键打开下拉菜单，选择新变量并确认后弹出跟踪配置，如图 8-46 所示。

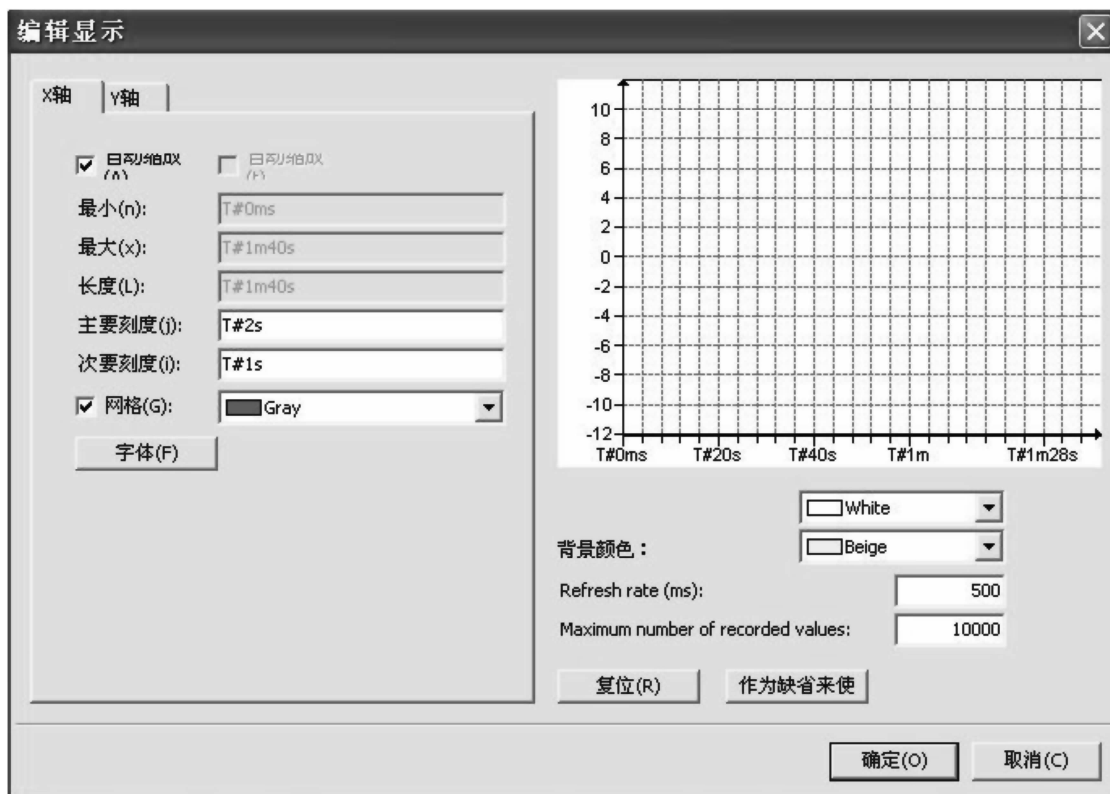


图 8-44　配置显示界面

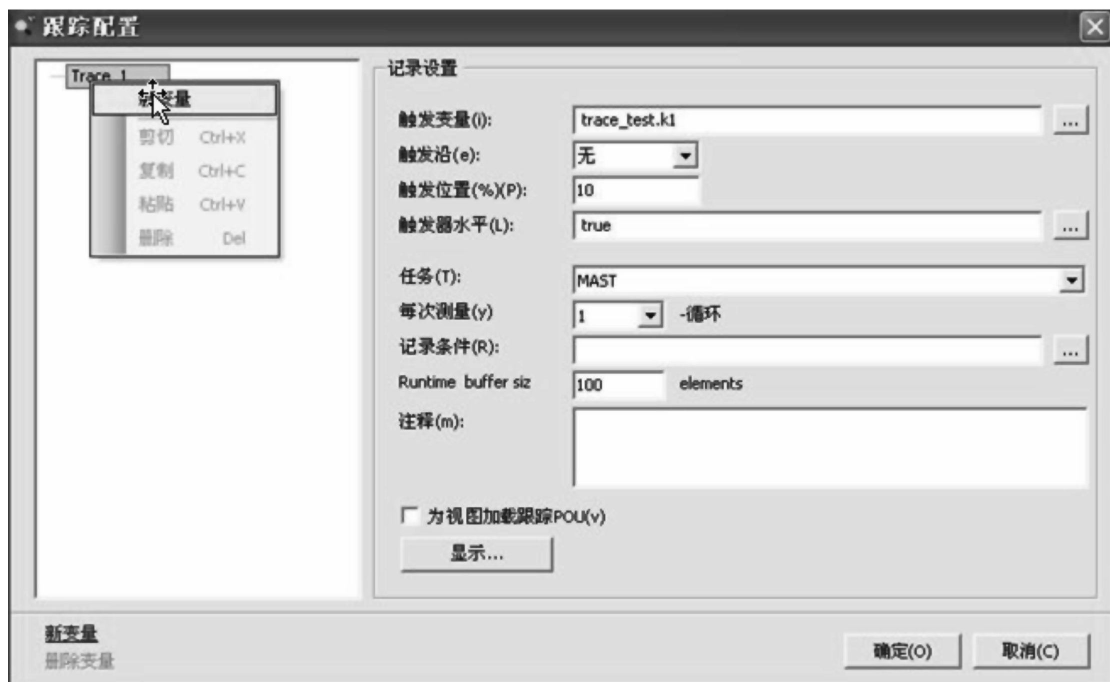


图 8-45　配置要记录的变量

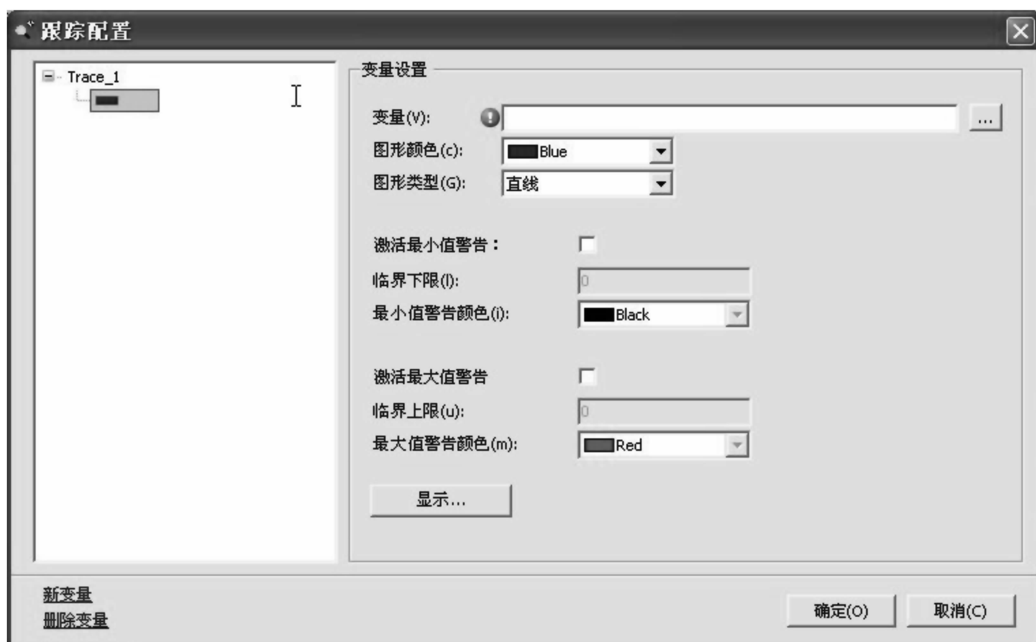


图 8-46　跟踪配置

点击变量栏目的右边图标，弹出输入助手如图 8-47 所示，选择要记录的程序变量 k1。

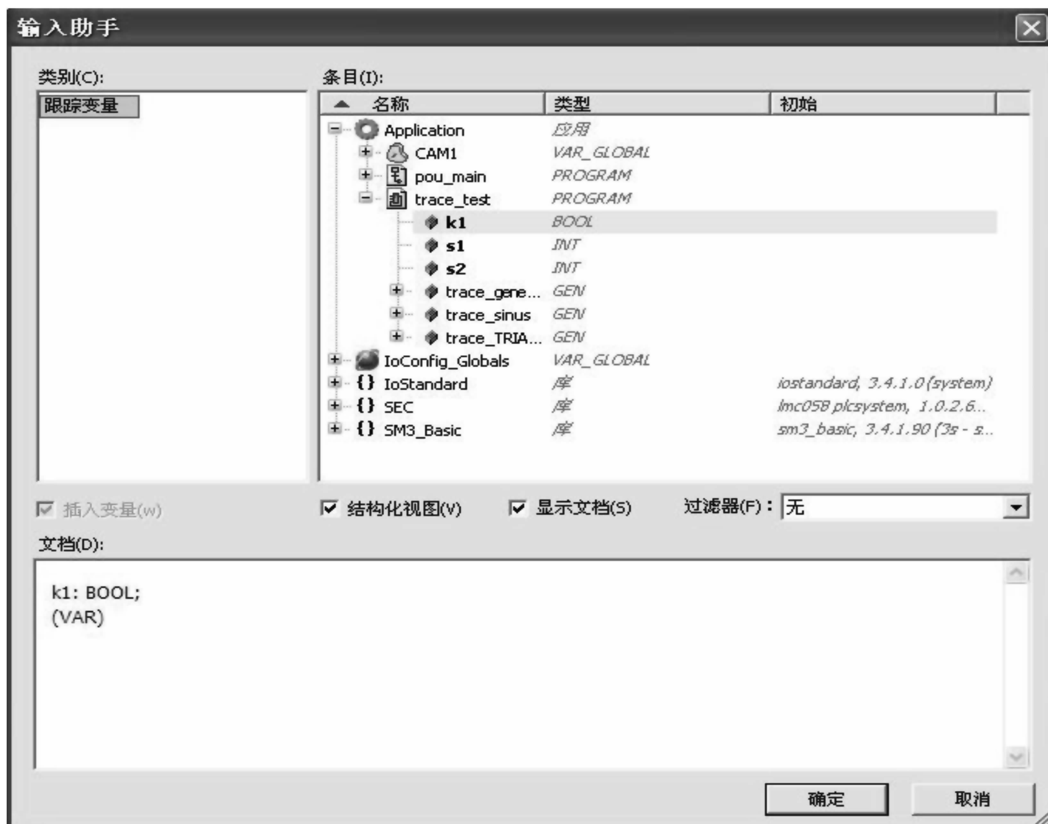


图 8-47　变量选择

确认后，变量 k1 被放入变量栏目中，如图 8-48 所示。

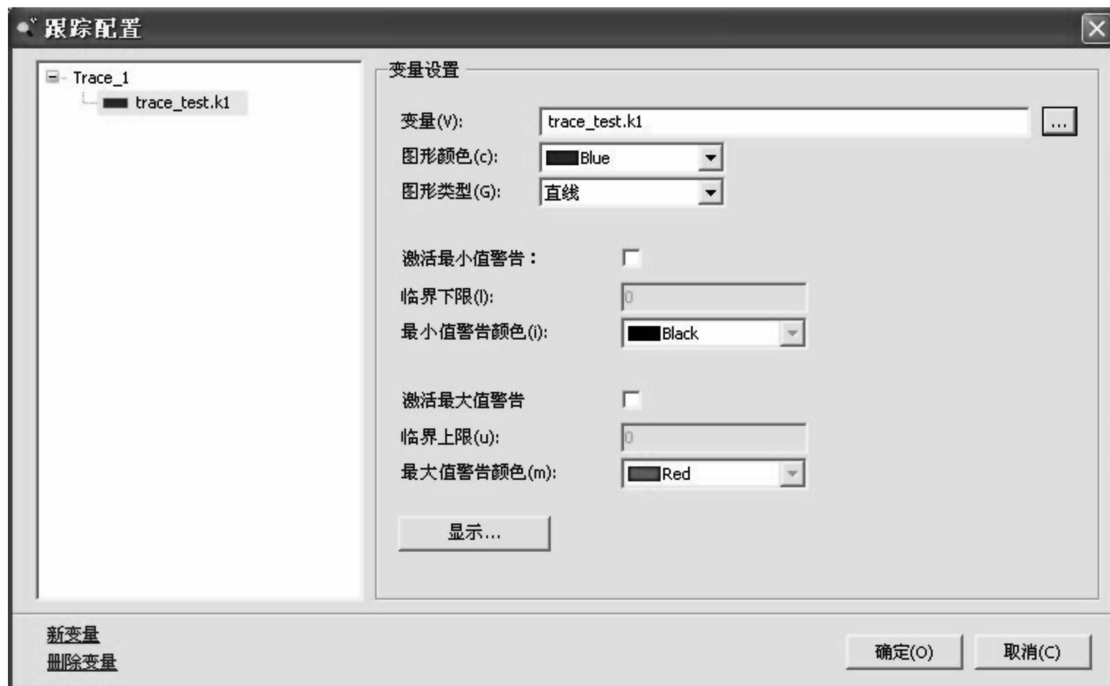


图 8-48　变量导入

同理，重复上述建立新变量，借助输入助手导入变量步骤，建立其他变量 s1 、s2。如图 8-49 所示。我们建立了 3 个变量的记录。



图 8-49　建立 3 个变量的记录

按“确定”后，建立的跟踪界面如图 8-50 所示。

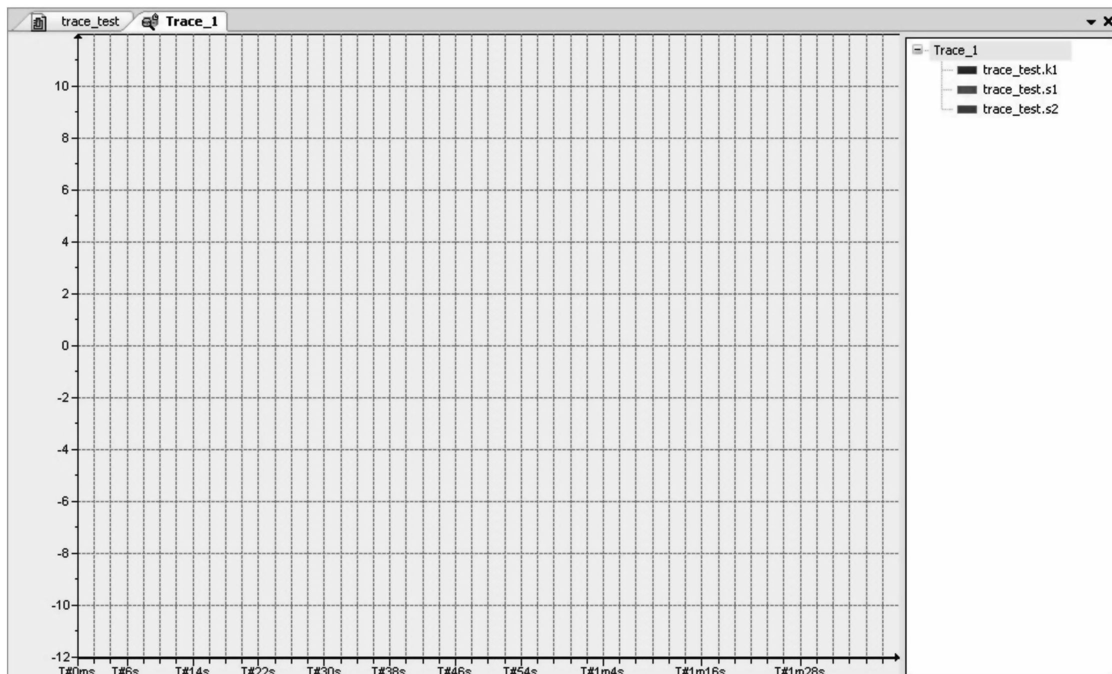


图 8-50　跟踪画面

开始仿真，首先使 k1 为“真”，这样曲线没有输出，如图 8-51 所示。

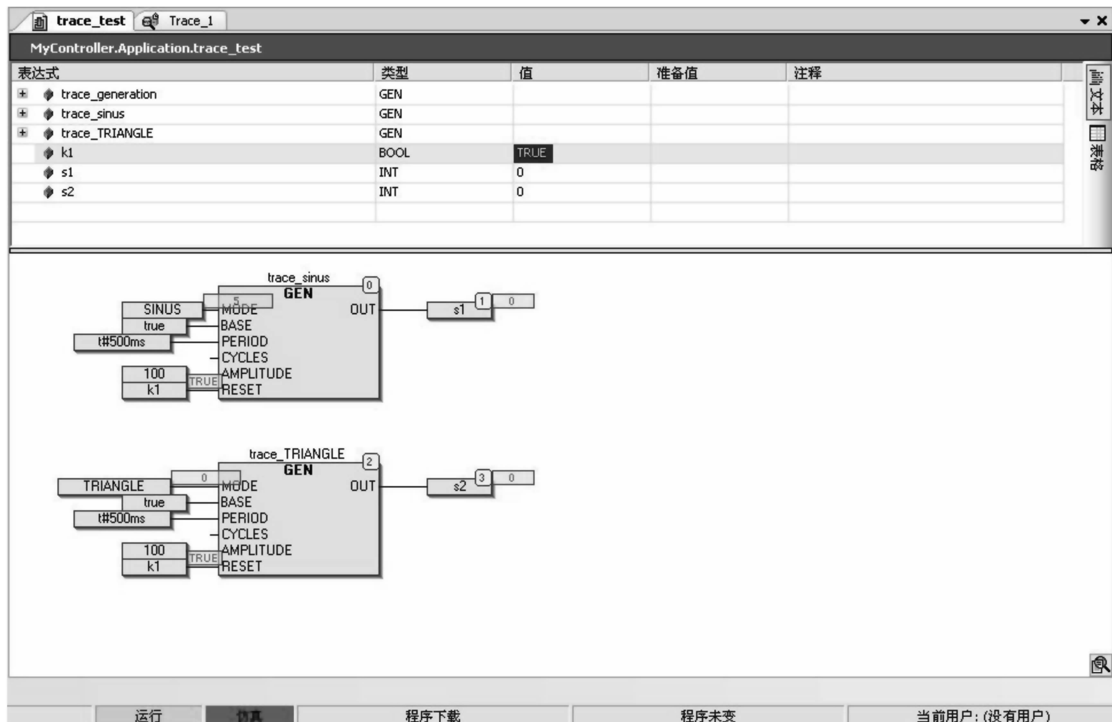


图 8-51　程序仿真执行

然后在图 8-52 所示的跟踪配置中，选择记录触发信号的触发沿为负向，即当 k1 从“1”变为“0”时，触发记录开始。记录位置定为 90，即触发后的记录占这个记录的 90%，也就是说，触发开始的记录从整个记录的 10% 处开始。这可以从后面的记录曲线看出。



图 8-52　设置触发信号沿

在记录区域点击鼠标右键，选择下载跟踪，如图 8-53 所示。

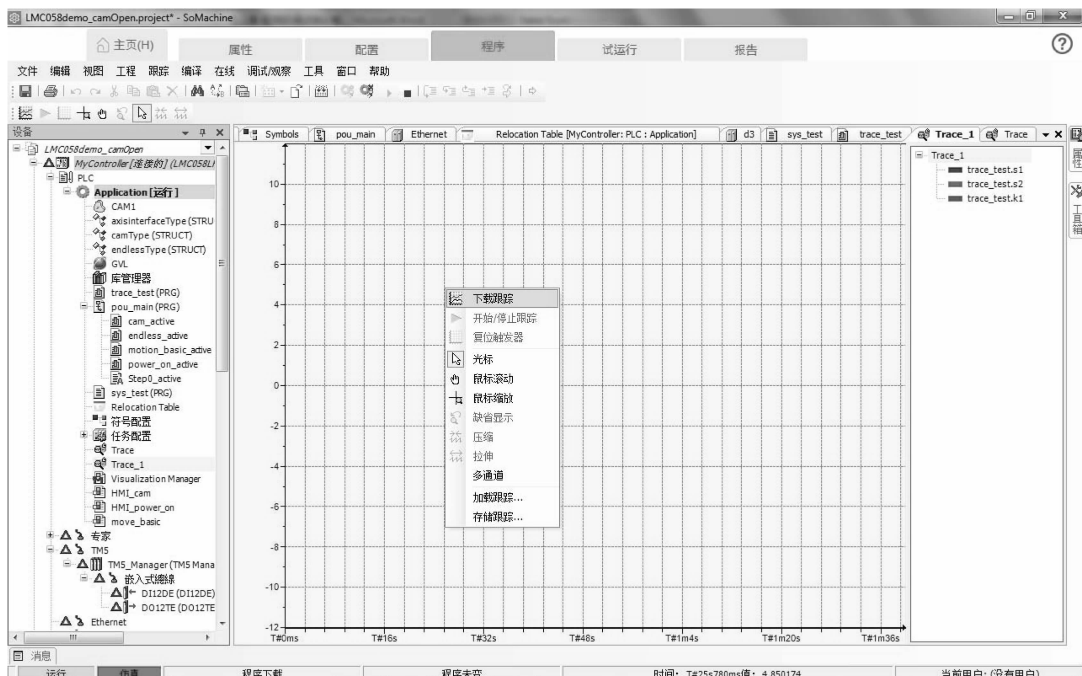


图 8-53　打开记录菜单选择下载跟踪

启动下载跟踪后，记录的开始取决于触发信号的到来。如图 8-54 所示，记录的启动等待着触发信号。

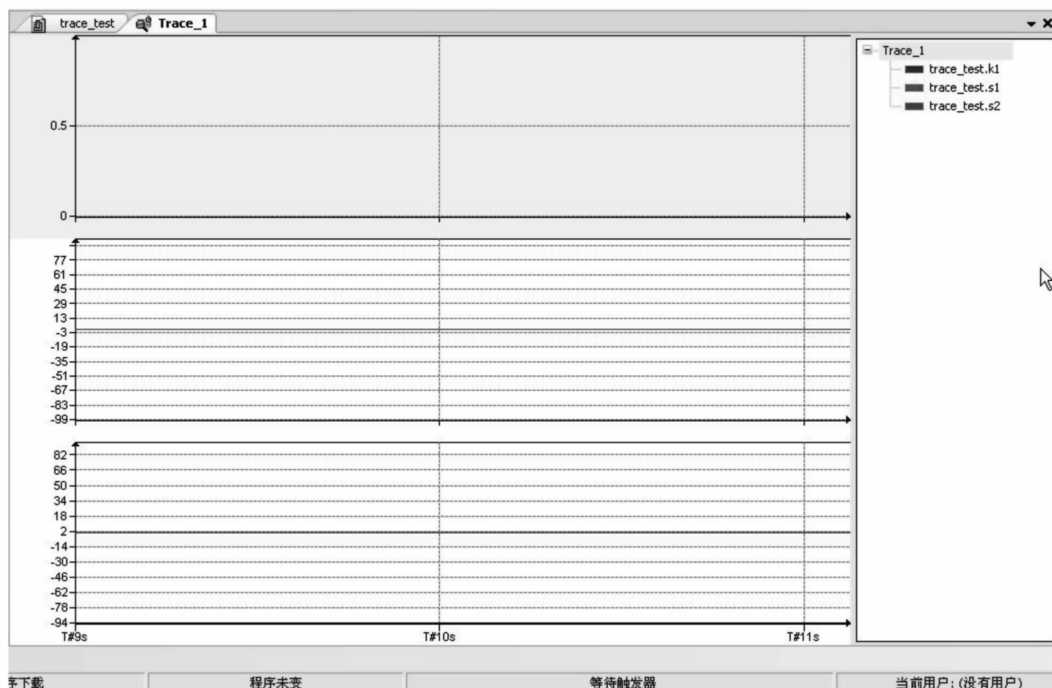


图 8-54　确认下载跟踪，记录等待触发信号

回到程序执行界面，使 k1 从 TRUE 变为 FALSE，如图 8-55 所示。从而启动了记录。

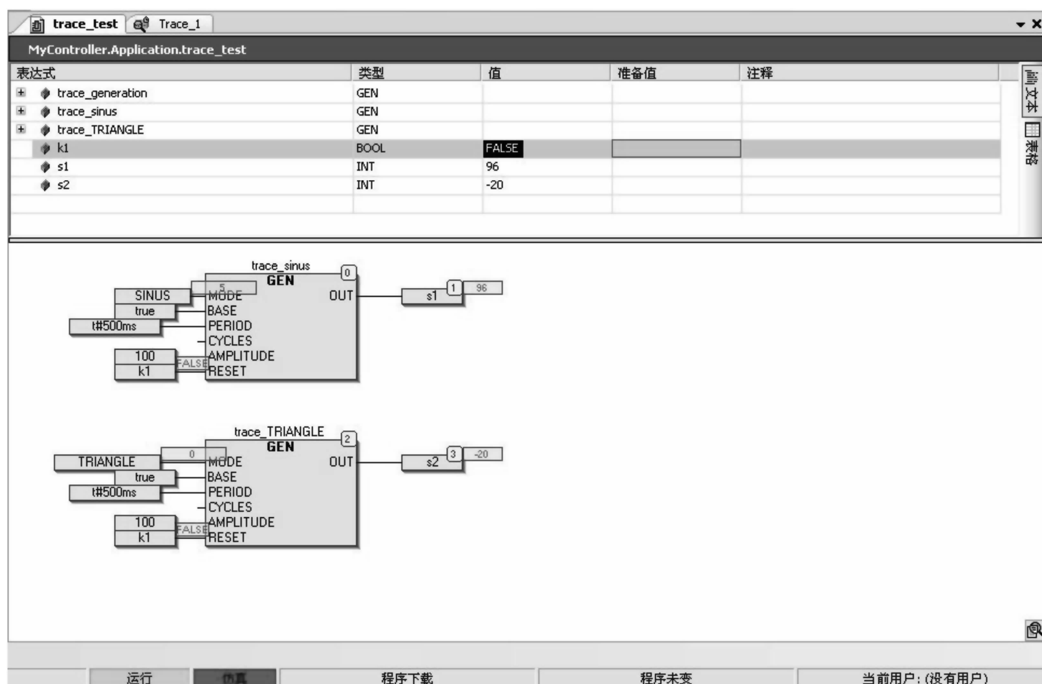


图 8-55　k1 从 TRUE 变为 FALSE

从图 8-56 看出，当 k1 从 1 变为 0 启动了曲线记录。这个触发点在这个记录画面的 10% 处，从这个触发时刻开始，曲线 s1 和 s2 输出，跟踪器 Trace _ 1 记录了触发后面的 90% 的运行情况。

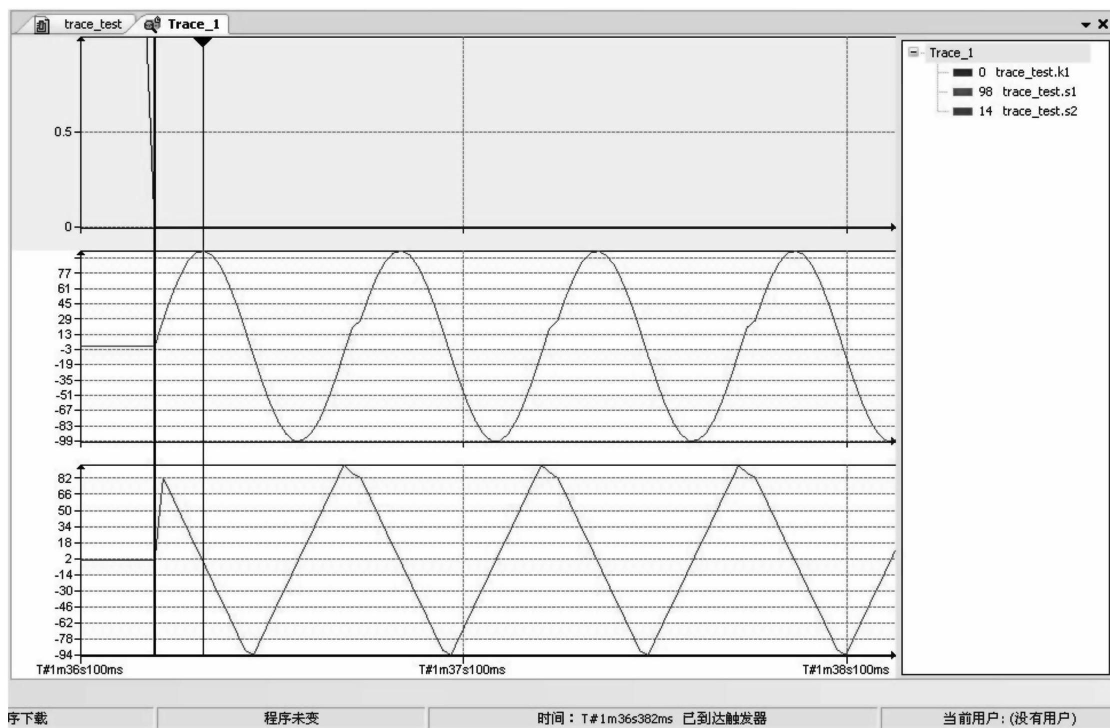


图 8-56　跟踪器记录的变量曲线

8.5　系统变量和功能

在每个控制器中，都有一些系统变量，通过对这些系统变量的读、写，我们可以了解这个控制器的状态和信息。打开在线帮助，可以看到可用的系统变量和功能说明，如图 8-57 所示。

在图 8-57 的标示 1 框中，我们知道通过系统变量的读出，可以了解控制器的固件版本和控制器状态等情况。也可以通过对系统变量的写操作来停止或启动控制器。还可以查看控制器的通信状态。

在标示 2 框中，我们看到控制器支持的一些附加功能，比如，立即 I/O 功能和冷启动/热启动探测功能。

在读取系统变量时，一种办法是：采用变量监视表，如图 8-58 所示。点击视图打开下拉菜单，建立一个监视表。

双击图 8-59 所示界面中表达式下空栏。这时空栏右边出现图标，点击图标，弹出输入助手。

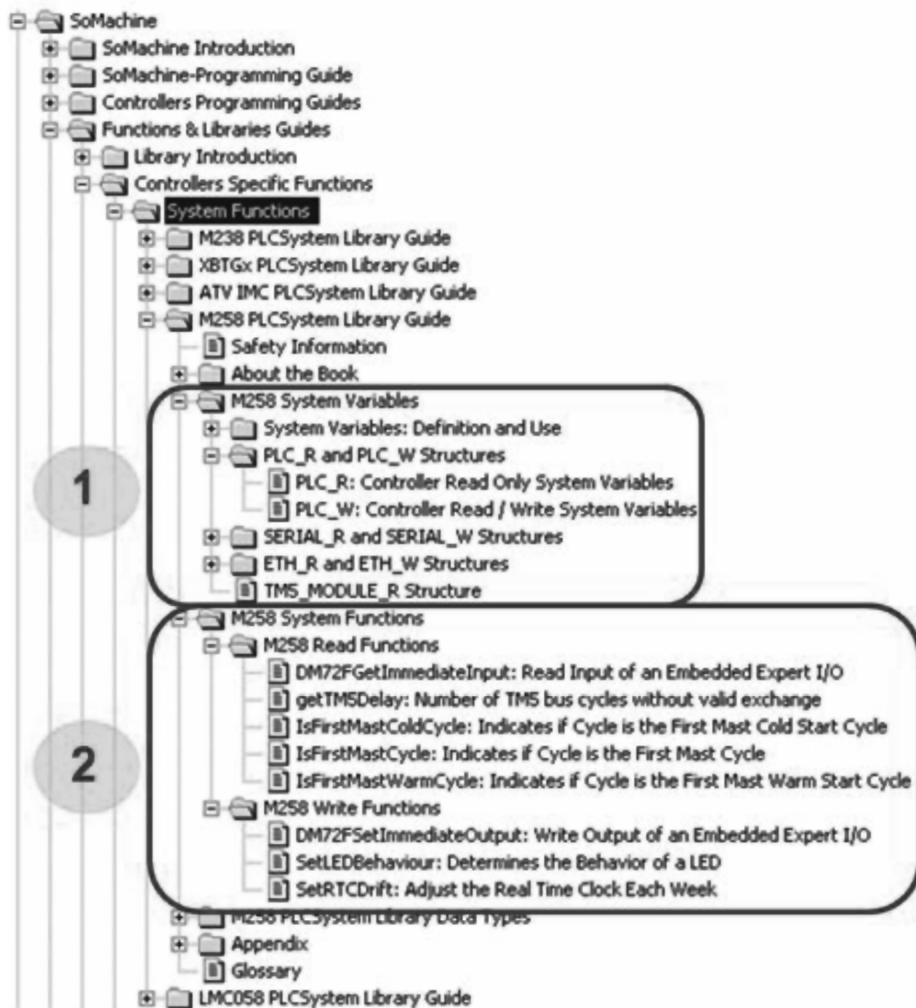


图 8-57　控制器的系统变量和功能

在输入助手中，按图逐层打开变量，选择 PLC _ R，按“确定”，如图 8-60 所示。这样监视变为如下界面，如图 8-61 所示。



图 8-58　建立监视表



图 8-59　配置监视表

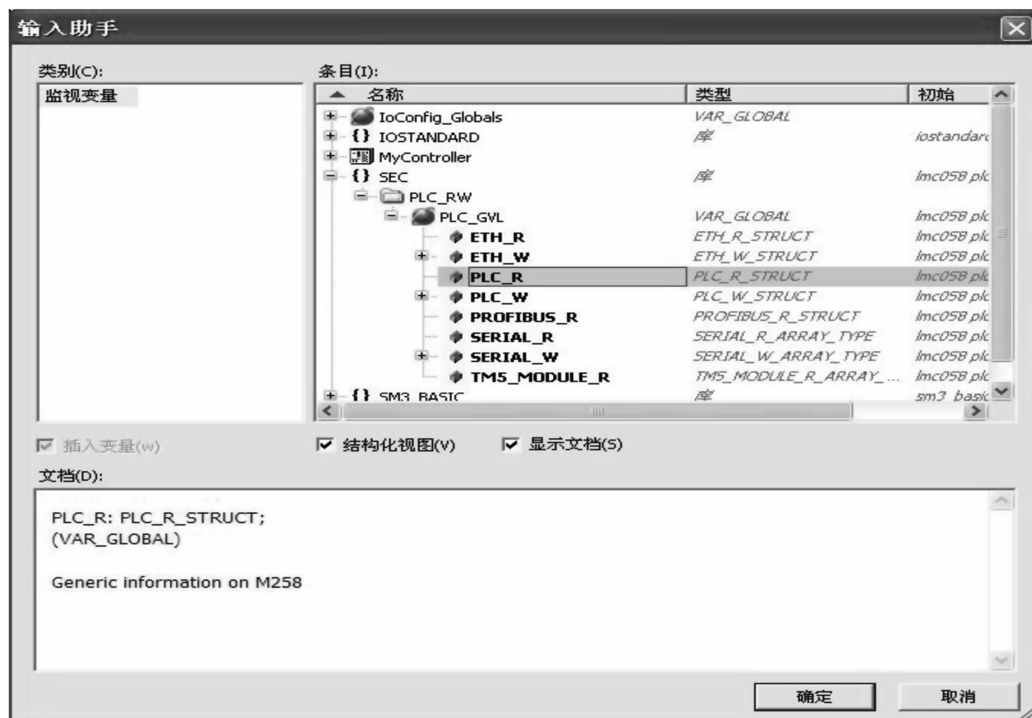


图 8-60　在输入助手中选择 PLC _ R

监视 1

表达式	类型	值	准备值	注释
MyController.Application.PLC_GVL.PLC_R	PLC_R_STRUCT			
i_wVendorID	WORD	<没有登录>		
i_wProductID	WORD	<没有登录>		
i_dwSerialNumber	DWORD	<没有登录>		
i_byFirmVersion	ARRAY [0..3] OF BYTE			
i_byFirmVersion[0]	BYTE	<没有登录>		
i_byFirmVersion[1]	BYTE	<没有登录>		
i_byFirmVersion[2]	BYTE	<没有登录>		
i_byFirmVersion[3]	BYTE	<没有登录>		
i_byBootVersion	ARRAY [0..3] OF BYTE			
i_byBootVersion[0]	BYTE	<没有登录>		
i_byBootVersion[1]	BYTE	<没有登录>		
i_byBootVersion[2]	BYTE	<没有登录>		
i_byBootVersion[3]	BYTE	<没有登录>		
i_dwHardVersion	DWORD	<没有登录>		
i_dwChipVersion	DWORD	<没有登录>		
i_wStatus	PLC_R_STATUS	<没有登录>		
i_wBootProjectStatus	PLC_R_BOOT_PROJECT_STATUS	<没有登录>		
i_wLastStopCause	PLC_R_STOP_CAUSE	<没有登录>		
i_wLastApplicationError	PLC_R_APPLICATION_ERROR	<没有登录>		
i_lwSystemFault_1	LWORD	<没有登录>		
i_lwSystemFault_2	LWORD	<没有登录>		
i_wIOStatus1	PLC_R_IO_STATUS	<没有登录>		
i_wIOStatus2	PLC_R_IO_STATUS	<没有登录>		
i_wClockBatterystatus	WORD	<没有登录>		
i_dwAppliSignature1	DWORD	<没有登录>		
i_dwAppliSignature2	DWORD	<没有登录>		
i_dwAppliSignature3	DWORD	<没有登录>		
i_dwAppliSignature4	DWORD	<没有登录>		
i_sVendorName	STRING(31)	<没有登录>		

图 8-61　监视画面出现可读出的控制器信息

登录控制器，我们可读到控制器的信息，如图 8-62 所示。

监视 1

表达式	类型	值	准备值	注释
MyController.Application.PLC_...	PLC_R_STRUCT			
i_wVendorID	WORD	0		
i_wProductID	WORD	0		
i_dwSerialNumber	DWORD	0		
i_byFirmVersion	ARRAY [0..3] OF BYTE			
i_byFirmVersion[0]	BYTE	0		
i_byFirmVersion[1]	BYTE	0		
i_byFirmVersion[2]	BYTE	0		
i_byFirmVersion[3]	BYTE	0		
i_byBootVersion	ARRAY [0..3] OF BYTE			
i_byBootVersion[0]	BYTE	0		
i_byBootVersion[1]	BYTE	0		
i_byBootVersion[2]	BYTE	0		
i_byBootVersion[3]	BYTE	0		
i_dwHardVersion	DWORD	0		
i_dwChipVersion	DWORD	0		
i_wStatus	PLC_R_STATUS	PLC_R_EMPTY		
i_wBootProjectStatus	PLC_R_BOOT_PROJECT_STATUS	PLC_R_NO_BOOT_PROJECT		
i_wLastStopCause	PLC_R_STOP_CAUSE	PLC_R_STOP_REASON_UNKNOWN		
i_wLastApplicationError	PLC_R_APPLICATION_ERROR	PLC_R_APP_ERR_APPLICATION_VERSION_MISMATCH		
i_lwSystemFault_1	LWORD	0		
i_lwSystemFault_2	LWORD	0		
i_wIOStatus1	PLC_R_IO_STATUS	0		
i_wIOStatus2	PLC_R_IO_STATUS	0		
i_wClockBatterystatus	WORD	0		
i_dwAppliSignature1	DWORD	0		
i_dwAppliSignature2	DWORD	0		
i_dwAppliSignature3	DWORD	0		
i_dwAppliSignature4	DWORD	0		
i_sVendorName	STRING(31)	''		

运行　仿真　程序下载　程序未变　当前用户：(没有用户)

图 8-62　控制器系统信息

还有一种办法是把系统信息赋值给一个变量，然后在应用程序中观察这个变量即可，如图 8-63 所示。

同理，可以用监视表来对控制器写操作，如图 8-64 所示。

控制器内的系统位“q _ wPLCControl”可以控制 PLC 的状态，支持的命令包含

"RUN"、"STOP"、"RESET _ COLD"、"RESET _ WARM"。当系统位"q _ uiOpenPLCControl"中间的数值由 0 变为 6699 时，将执行"q _ wPLCControl"内的控制命令。



图 8-63　把系统变量赋值给其他变量

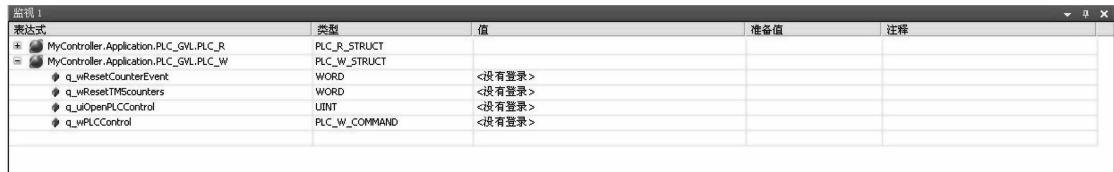


图 8-64　系统变量通过监视表对控制器写操作

系统变量通过赋值进行写操作如图 8-65 所示。

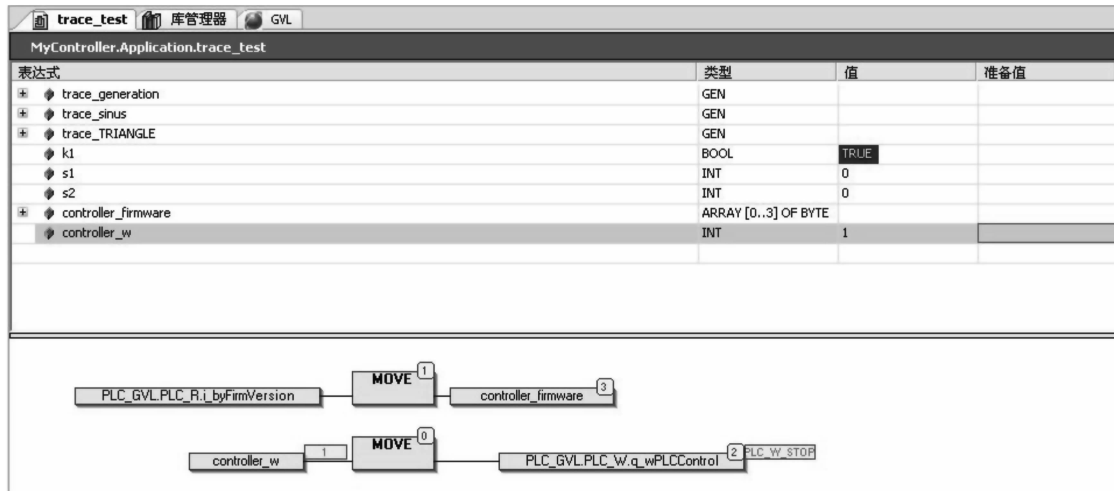


图 8-65　系统变量通过赋值进行写操作

第 9 章　可视界面的创建及应用

在编辑一些复杂程序时，我们对程序或功能块的执行结果还没有一定的把握，这时我们可以先不用编辑复杂的程序，而可以使用可视界面来操作功能块或核心程序，使用可视界面来观察执行状态，达到验证操作规程的目的。一旦我们的程序和功能块的操作规程被验证无误，就可以把可视界面的操作过程编成程序，从而达到事半功倍的效果。在这方面 SoMachine 编程软件都设计了很方便的编辑工具和专用模板供使用。

9.1　SoMachine 编程软件集成的可视界面

采用 SoMachine 图形编辑器，可以把内部变量的数值动态地显示在界面中，如图 9-1 所示。通过这些变量在图形中的变化，就能直观地观察到程序运行的状况。

SoMachine 编程软件还提供了一些专用的图形模板，例如 PLCopen 的可视化界面，如图 9-2 所示。

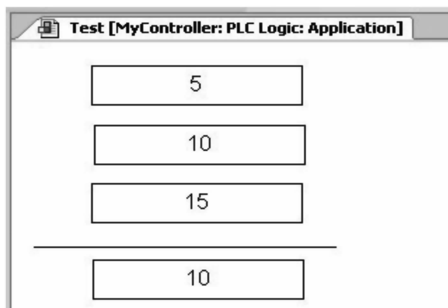


图 9-1　可视界面的应用

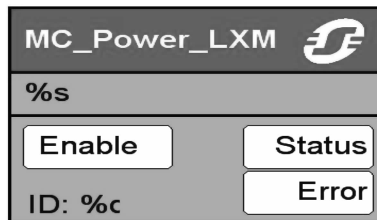


图 9-2　运动控制中驱动的使能模板

这些界面的编辑，我们都可以通过 SoMachine 编程软件集成的编辑工具来完成。这些工具包括工具箱（ToolBox），如图 9-3 所示。

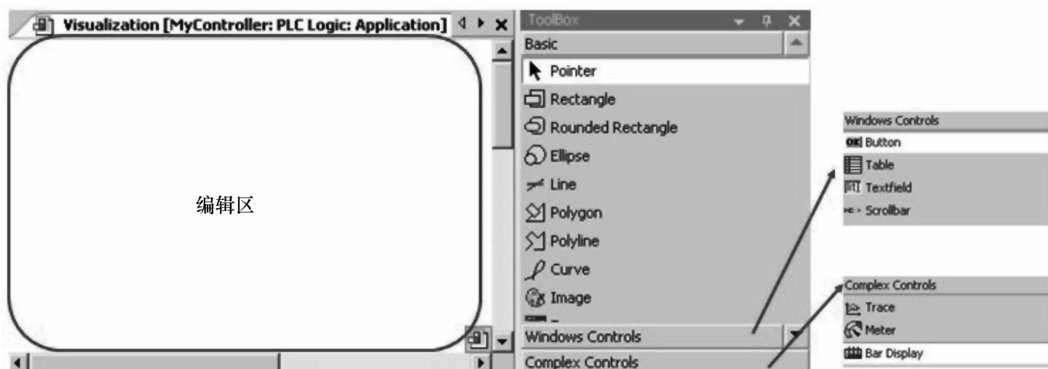


图 9-3　工具箱菜单

在工具箱界面，你可以看到分为 3 个类别：一个是基本型 Basic；一个是视窗控制 WindowsControls；一个是复杂控制 ComplexControls。我们可以用鼠标点击这 3 个类别下的任何一个元素，把它拖拽到编辑区进行编辑。当把这些元素拖拽到编辑区，我们就可以打开这个元素的属性编辑框来编辑元素的各种属性了，如图 9-4 所示。

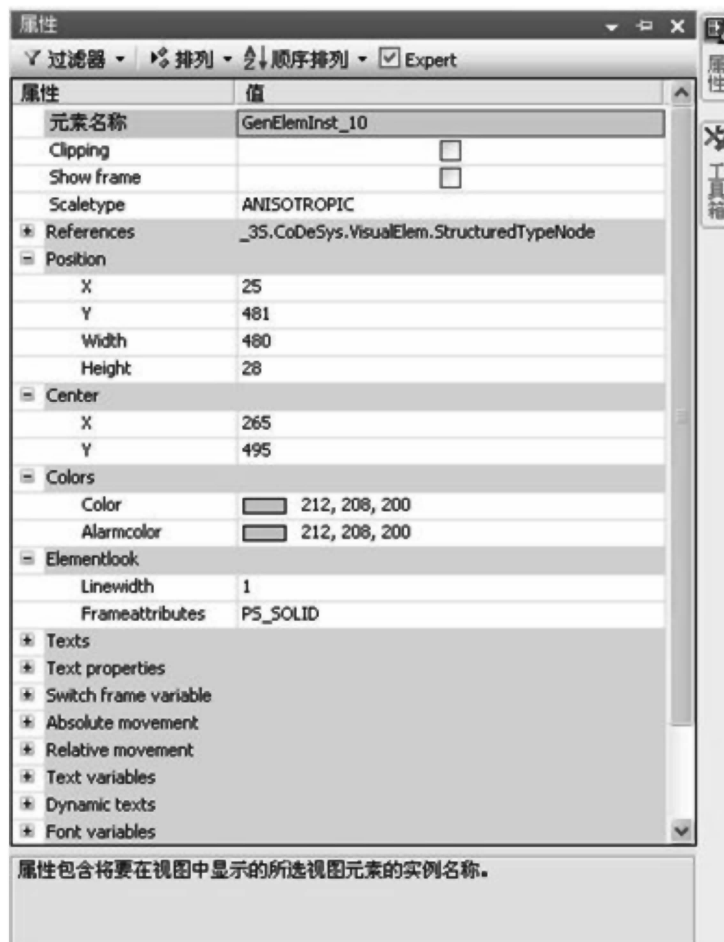


图 9-4　图形元素的属性

通过对这个元素的属性编辑，我们就可以把这个元素用于：

- 观察；
- 内部动作输出（例如：改变颜色，显示数值，…）；
- 运动；
- 可视图形；
- 内部动作输入（例如：改变变量值，写数据，…）。

9.2　视图的建立及编辑

1. 编辑一个开关量

在如图 9-5 所示程序结构中，我们把 k1 编辑到可视界面上。通过对 k1 的操作，可以对

程序流进行控制。

在设备菜单，在应用 Application 处，点击鼠标右键打开添加对象-视图，如图 9-6 所示。

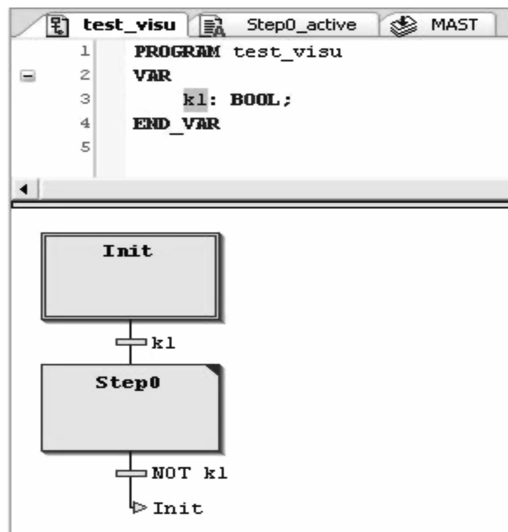


图 9-5　程序结构中 k1 的动作

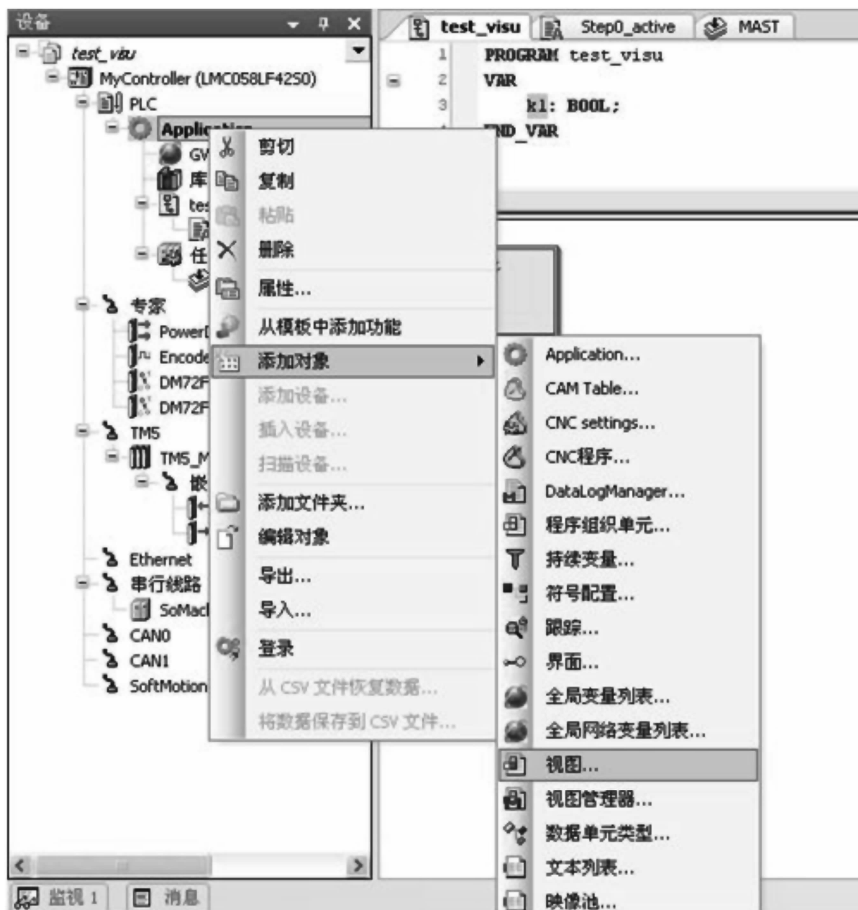


图 9-6　添加视图

点击视图，打开如下界面，如图 9-7 所示。



图 9-7　创建视图

给视图命名，然后点击打开，出现如图 9-8 所示编辑界面。

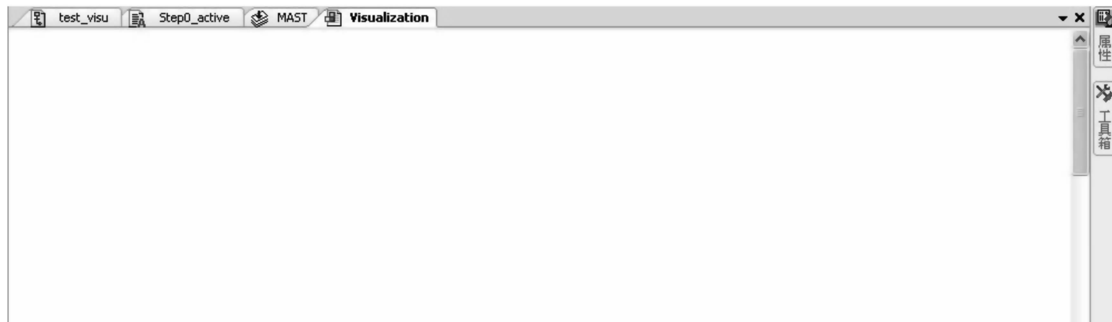


图 9-8　视图画面编辑区

点击右侧工具箱，在打开的工具箱栏目下，选择一个画圆图标 Ellipse，如图 9-9 所示。

按住鼠标左键，把选中的图标拖到视图编辑区内放开，如图 9-10 所示。

选中图标元素，然后打开属性框，如图 9-11 所示。在属性框中，点开色彩栏 Colors，在 Fillcolor 框内颜色菜单中选择需要的色彩，例如：红色；在报警状态栏 Alarmstate 的框内颜色菜单中选择所需要的颜色，例如：绿色；在文本栏目 Texts 写入要控制的开关 k1，如图 9-12 所示。



图 9-9　打开工具箱选中图标

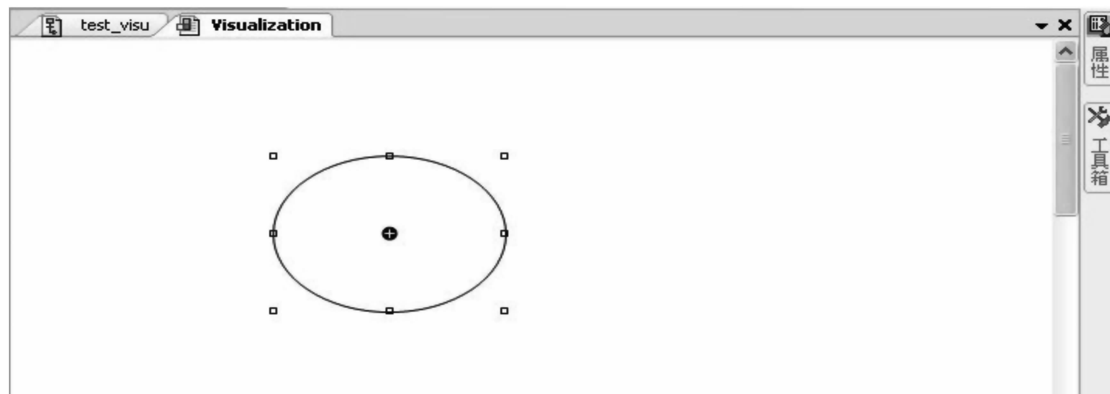


图 9-10　把编辑元素拖到编辑区

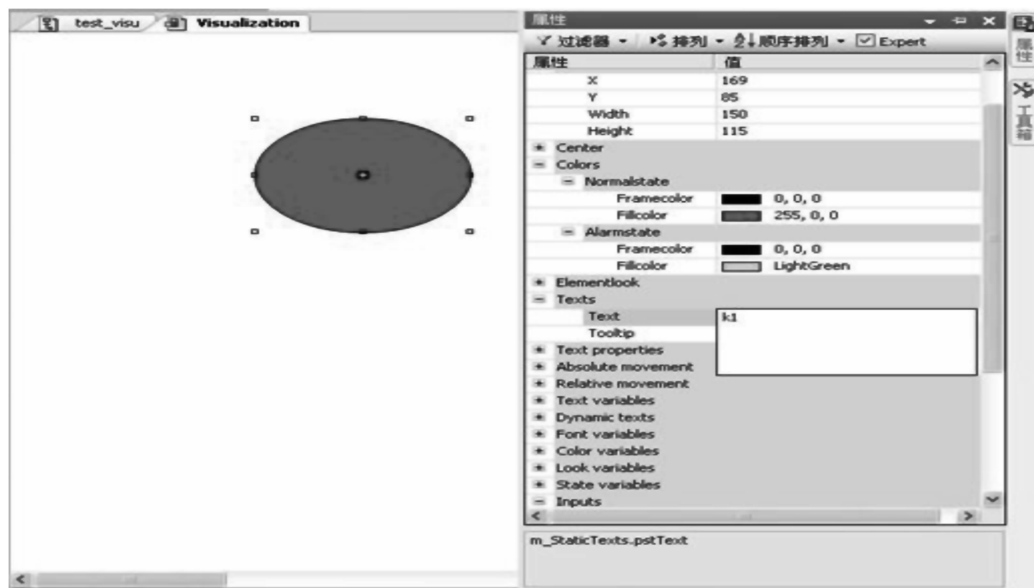


图 9-11　打开元素属性，填写属性内容

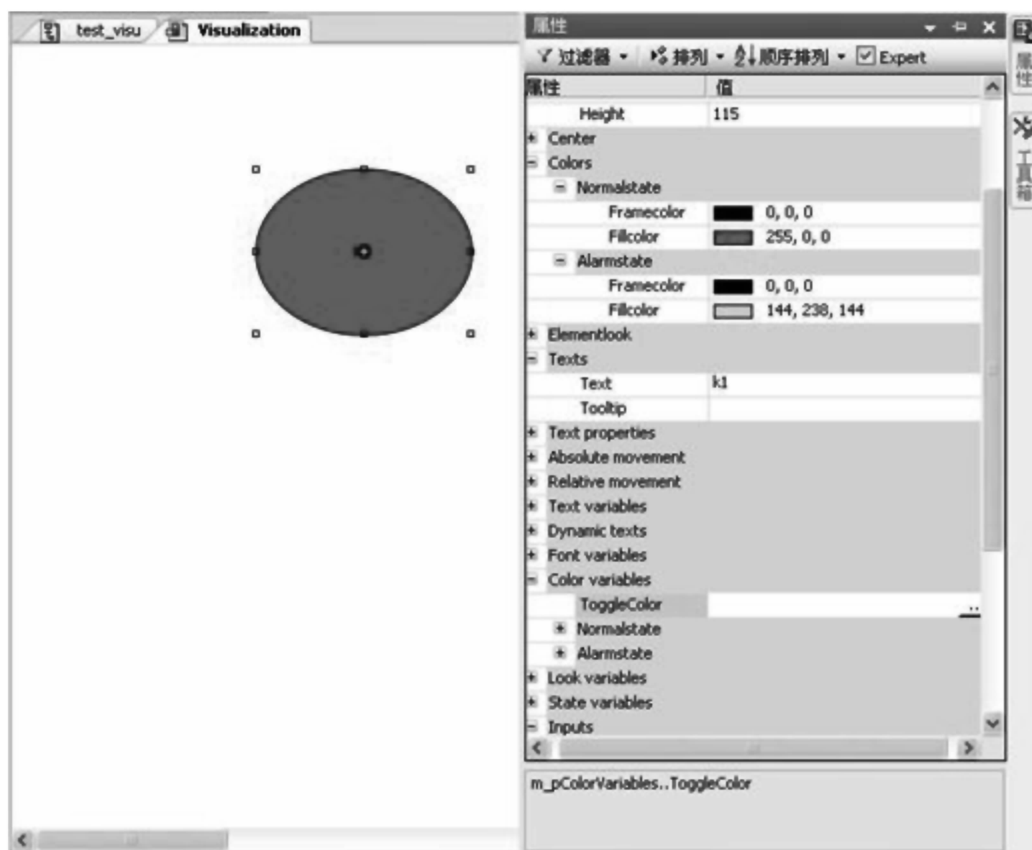


图 9-12　打开元素属性，填写改变色彩的变量

如图 9-13 所示，打开颜色变量栏目，选择“ToggleColor”，点击右侧助手框，打开输入助手。

在输入助手框中，选择改变颜色的变量 k1，并按“确定”。

同理，点开输入“Inputs”，填写所编图标的输入要改变的变量 k1，如图 9-14 所示。

这样，我们就完成了对一个开关 k1 的编辑。我们可以用仿真来测试一下。

打开程序仿真并运行，如图 9-15 所示，这时程序处于初始化阶段。

点击图标 k1，则使图标变绿，即使能了 k1，如图 9-16 所示。这时程序扫描进入 Step0 阶段。通过这个开关，我们看到了对程序流的控制。

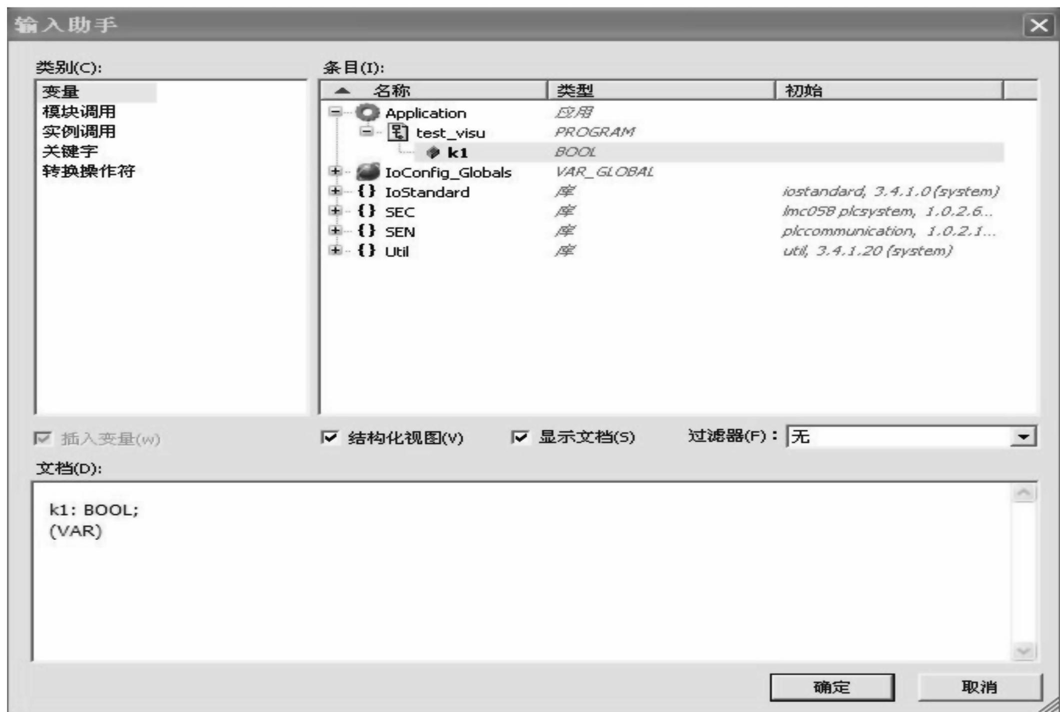


图 9-13　选择改变颜色的变量 k1

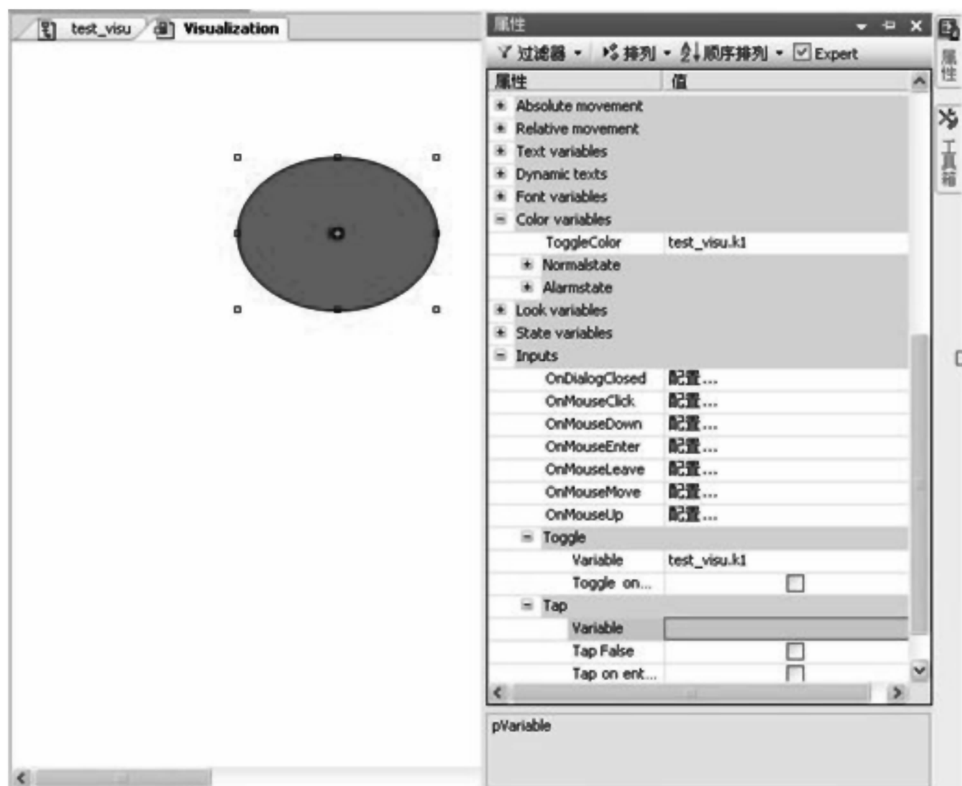


图 9-14　确定后，变量填入“ToggleColor”和“ToggleVariable”栏目



图 9-15　开关 k1 处于打开状态

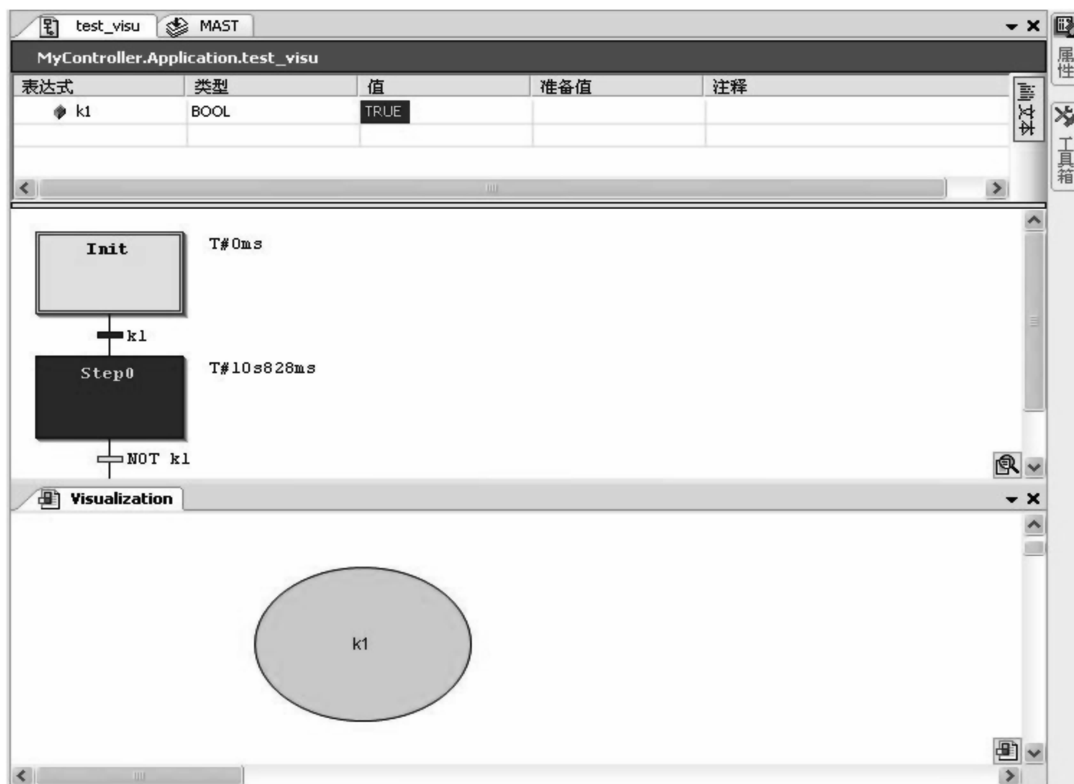


图 9-16　开关 k1 处于关闭状态

2. 编辑一个显示和输入界面

在程序 Step0 阶段，调上一个曲线发生器。我们设计一个界面，通过界面可以对曲线的幅值 amp 进行修改，同时通过界面观察输出 s1 的变化。视窗和程序结构如图 9-17 所示。

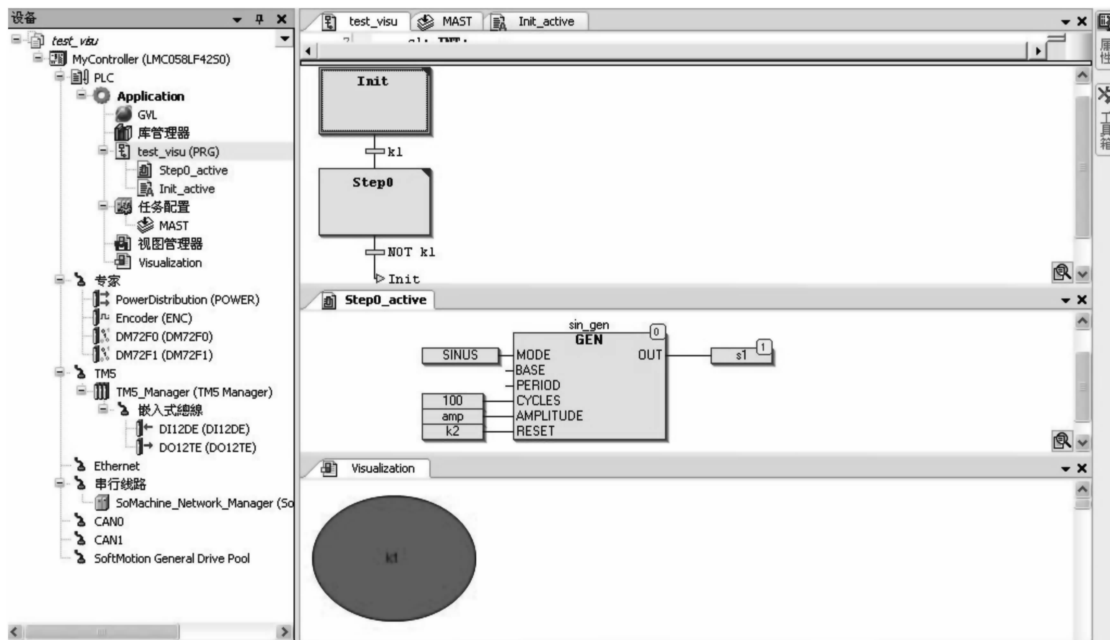


图 9-17　视窗和程序结构

在视窗中，我们可以再重复一下建立 k1 的过程来建立开关 k2，也可以采用复制-粘贴 k1 的方式来建立 k2。复制-粘贴的方式如下：点选视窗中的 k1，按“Ctrl + C”即复制快捷键，然后再按“Ctrl + V”即粘贴快捷键。这样在视窗中就复制了 k1，如图 9-18 所示。

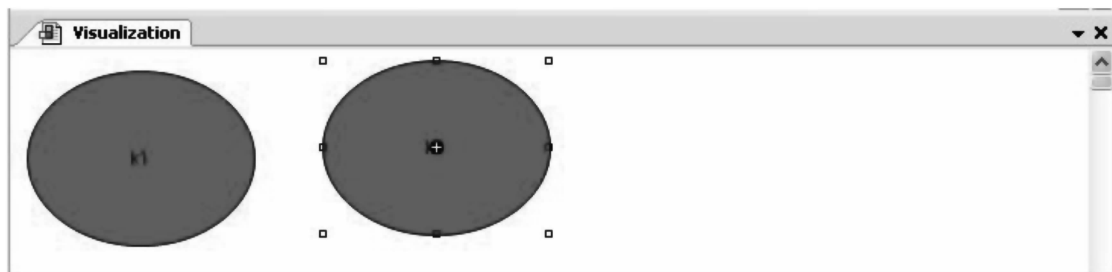


图 9-18　在视窗中复制 k1

然后打开图形的属性，修改文字为 k2，修改引起颜色变化的变量 k2 和输入导致变化的变量 k2，如图 9-19 所示。

打开工具箱，选择一个长方形图标 Rounded Rectangle ，把它放置在视图界面编辑区，如图 9-20 所示。

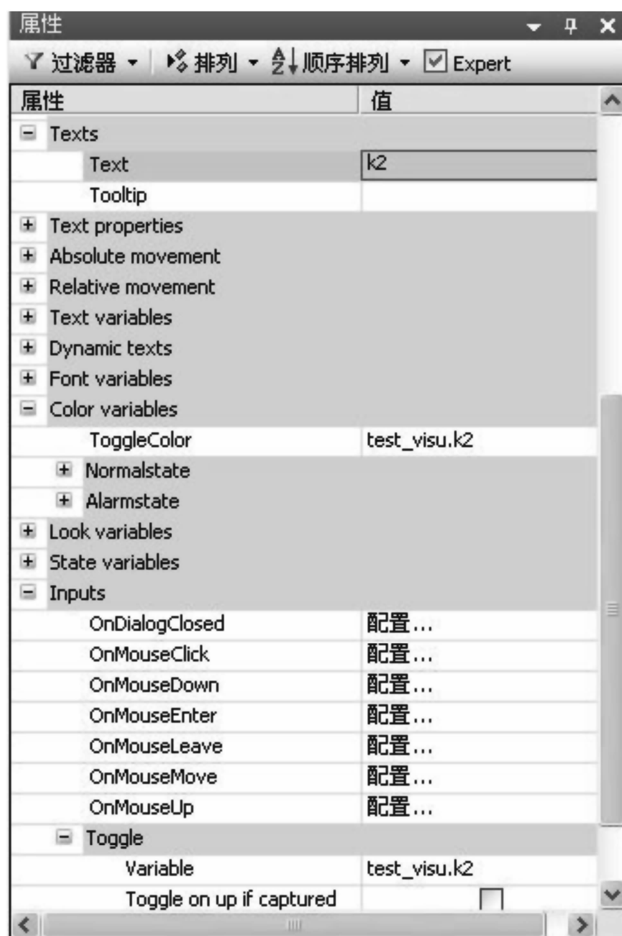


图 9-19　修改元素属性

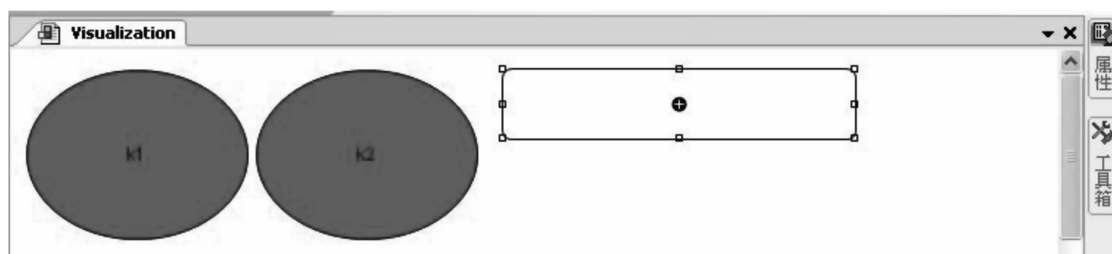


图 9-20　打开工具箱调入一个矩形框

然后，点击“属性”，打开属性框，建立一个曲线输出 s1 的显示。属性的填写如图 9-21 所示。在颜色栏，选择框内背景颜色为黄色。在文字栏，写入“s1 %s”，百分号前是显示名，百分号后是字符串输出，即 s 表示字符串 String，注意百分号后的字符要小写。当然，如果要显示其他类型的变量，则百分号后的字符是不一样的。如常用的%f 浮点数;%0. 2f

浮点数并且保留小数点后 2 位;%d 或%i 十进制数;%b 二进制数等。

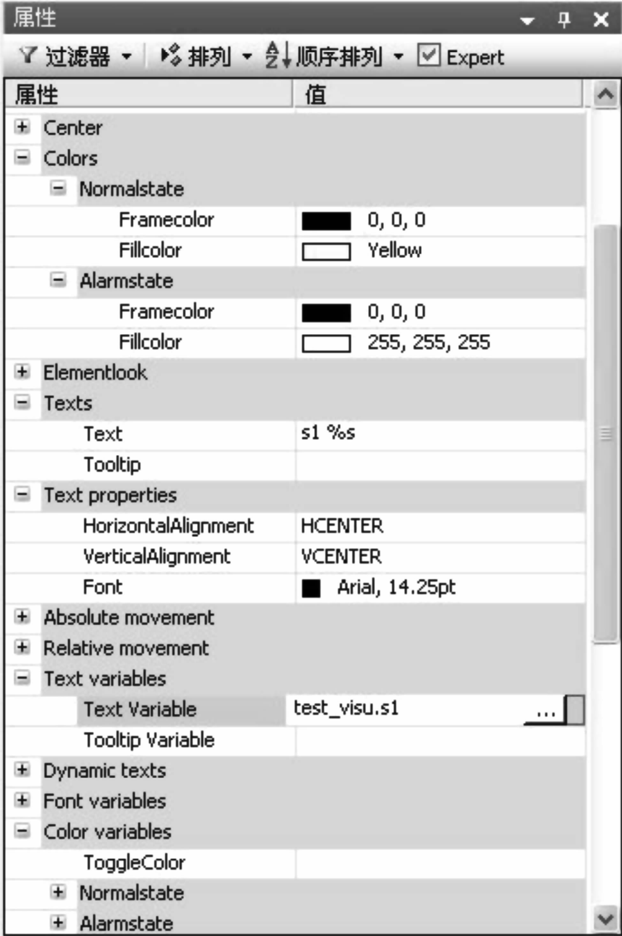


图 9-21　显示框属性的填写

在文本变量栏，填入要显示的变量。这个变量的写入也可通过点击右侧的输入帮手，来选择要显示的变量。例如选择“test _ visu. s1”。这样就完成了对变量 s1 的输出值变化显示，如图 9-22 所示。

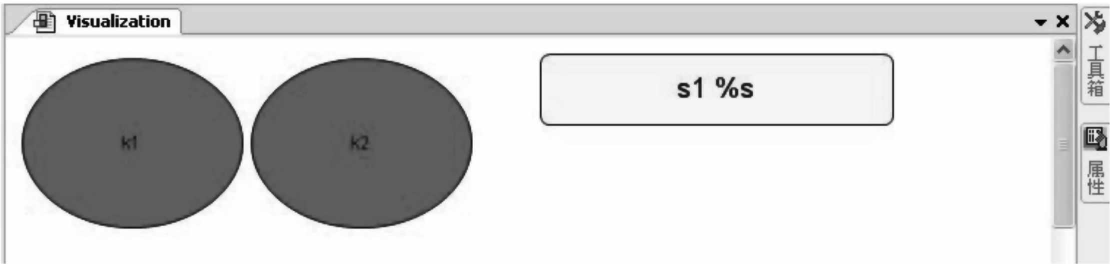


图 9-22　输出变量显示

仿真运行，如图 9-23 所示。

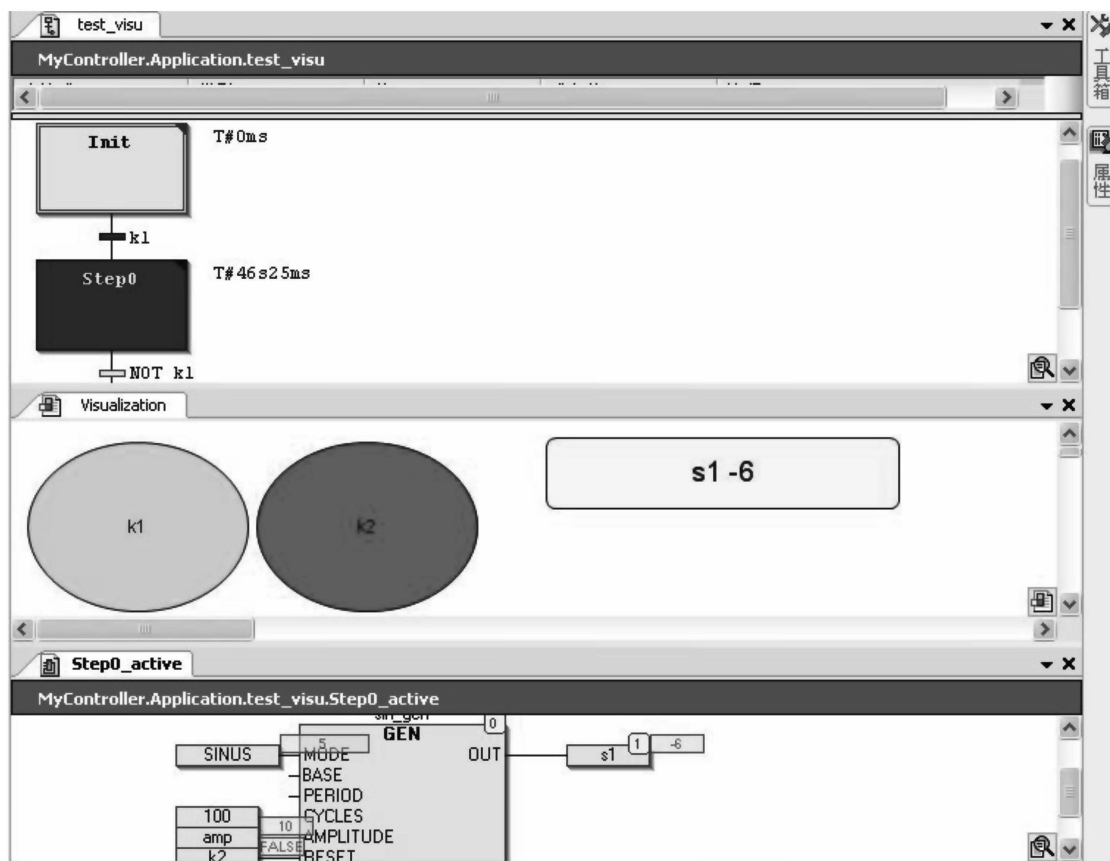


图 9-23　仿真运行观察到的 s1 情况

再次打开工具箱，选择一个长方形图标 Rounded Rectangle ，把它放置在视图界面编辑区，如图 9-24 所示。

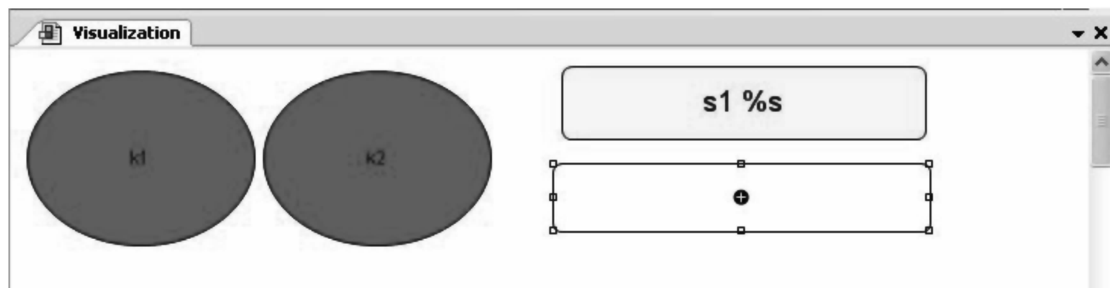


图 9-24　调入一个矩形框建立输入界面

打开相应属性框，建立一个对曲线幅值 amp 的赋值界面，如图 9-25 所示。在这个属性界面，我们在颜色 Colors 栏，选择背景色为黄色。在文本栏 Texts 填写 “amp:%i” 表示要输入的变量名是：amp，由于这个变量是 “INT” 类型，因此百分号后是 “i”。在文本变量

栏 TextVariable，点击右侧的输入助手，可在输入助手的变量栏目找到要赋值的变量，选定后，按“确定”，即自动把变量 test _ visu. amp 填入栏内。

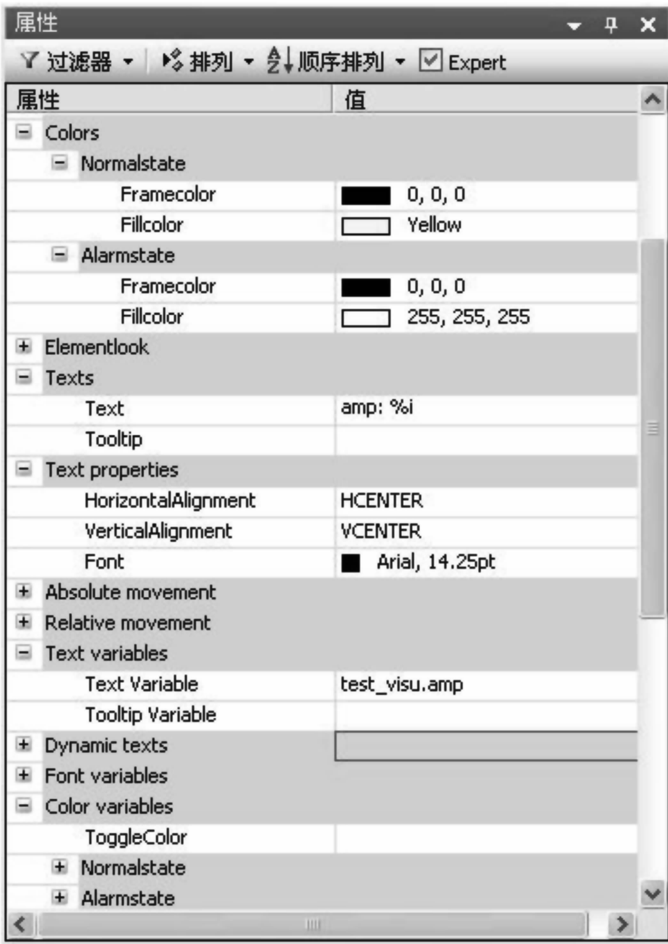


图 9-25　输入界面的属性

填好上述栏目后，作为输入功能，还要建立一个输入 Inputs。所以把属性菜单下拉，开启一个输入的配置。如图 9-26 所示，在输入部分，给出了输入的各种方式，根据需要可选择一种方式。例如：选择点击鼠标 OnMouseClick 配置…。

点击所要方式的配置栏，打开如图 9-27 所示界面。

在这个界面中，给出了输入动作的各种执行方式，我们选择“写变量”，即选中“写变量”，然后点 >，把它导入右侧，如图 9-28 所示。

在右侧写变量下的编辑类型栏，可以使用下拉菜单，选择可视数字对话板来输入变量数值。选择使用另一个变量后，激活下部的变量栏。通过点击右侧的输入助手，选取要赋值的变量。按“确定”后，完成对输入的配置组态，如图 9-29 所示。

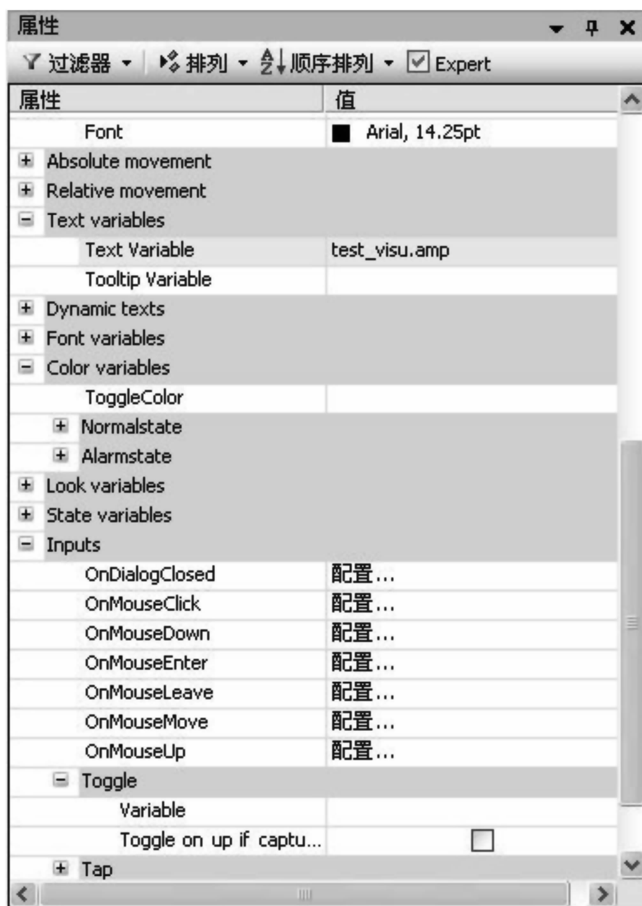


图 9-26　配置输入

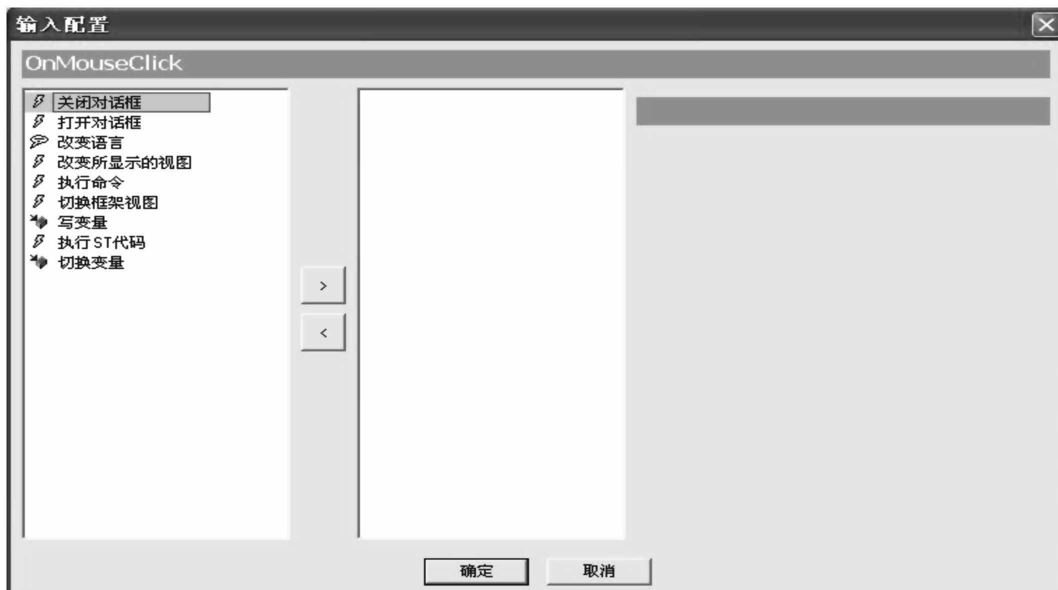


图 9-27　输入配置框



图 9-28　输入配置的组态



图 9-29　输入配置完毕

编辑完后的视图如图 9-30 所示。

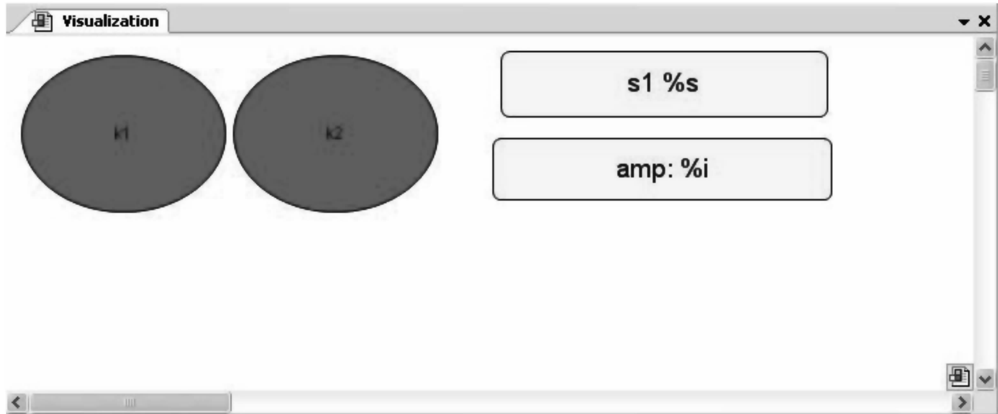


图 9-30　视图编辑完毕

没修改曲线幅值的仿真如图 9-31 所示。

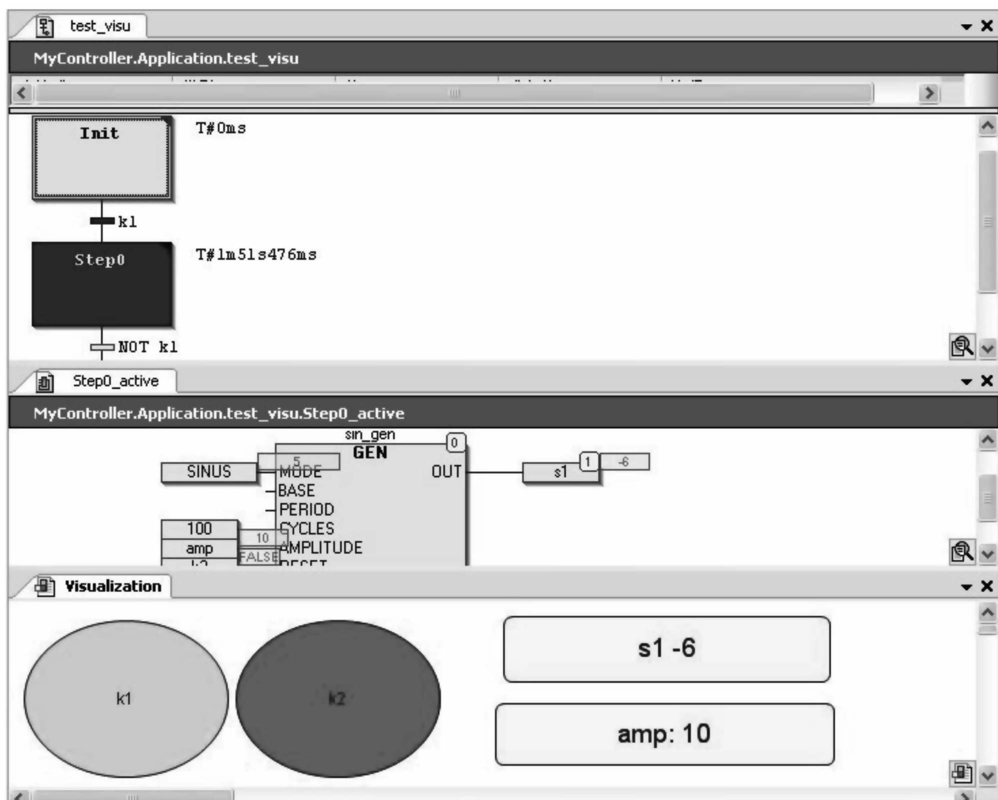


图 9-31　没修改曲线幅值的仿真

修改幅值时，点击 amp 输入界面中，则在这个视图中弹出一个数字输入面板，如图 9-32 所示。

利用这个输入面板，我们可以输入要修改的数值。比如，把幅值修改为 100。则在面板中输入 100，并点击“OK”即可。需要注意的是：有时这个面板显示的位置在整个视图的右下方，当视图的显示为 100%，即 1：1 时，看不到这个数字板。这时需要点击右下角的显示比例，把显示改为 75% 或 50%，这样就可以看到了。



图 9-32　数字输入面板

曲线幅值修改完后的仿真如图 9-33 所示。

我们也可以采用其他输入方式，例如采用执行 ST 代码方式，如图 9-34 所示。

采用执行 ST 代码方式，类似于编写一段程序，把数值或命令码赋值给变量。首先选定执行 ST 代码，然后点 >，把它导入右侧，点击右侧执行 ST 代码后，会出现执行 ST 代码的编程区，在这个编程区按照结构文本（ST）的编程语言编写程序即可。

完成新的输入配置后，进行的仿真如图 9-35 所示。这时的幅值 amp 为 10。

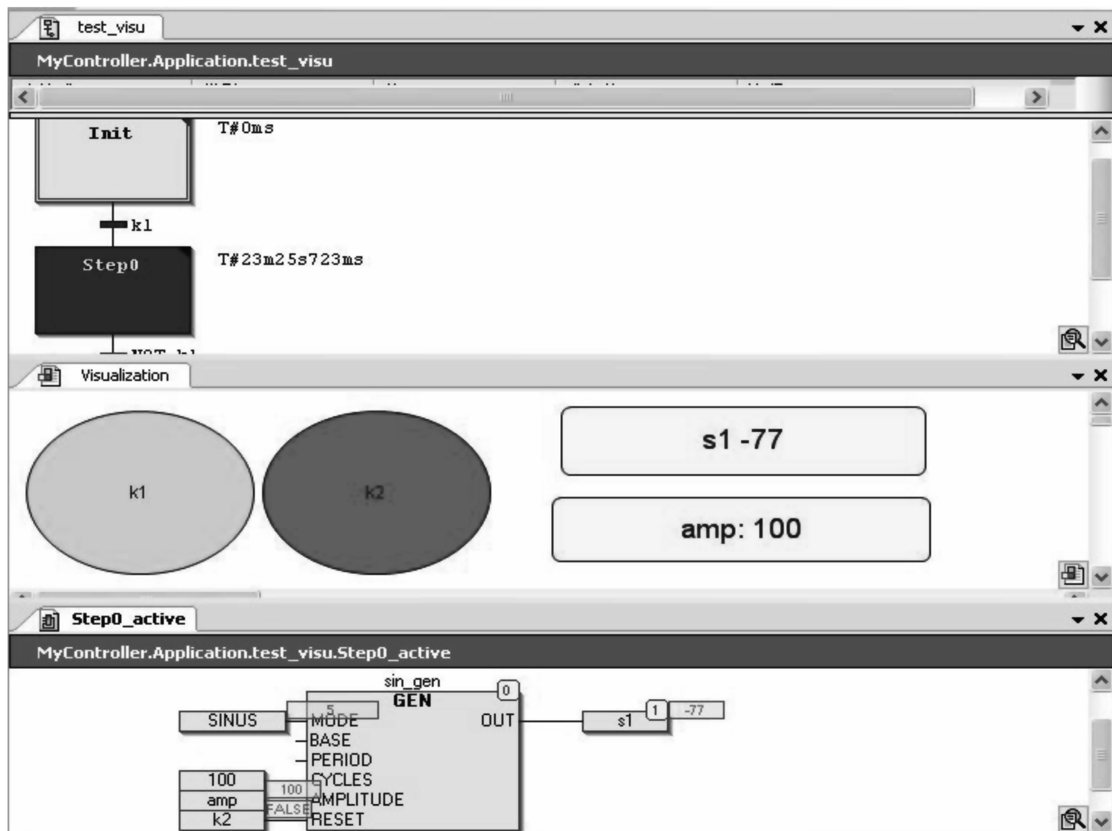


图 9-33　修改后的视图仿真

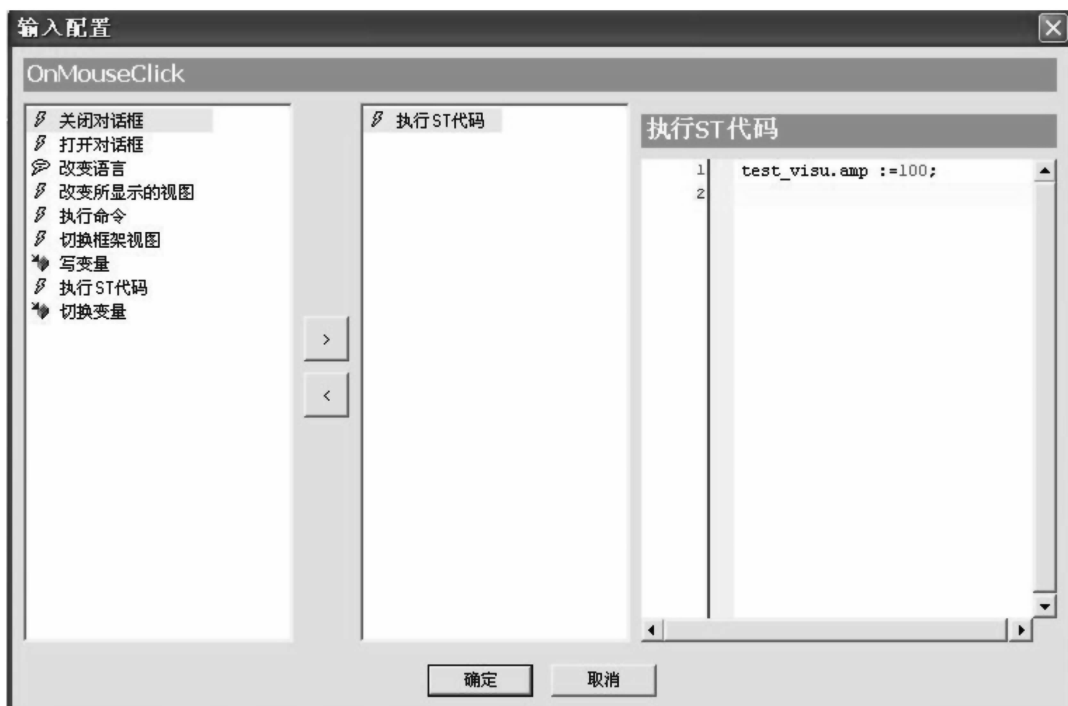


图 9-34　采用执行 ST 代码方式

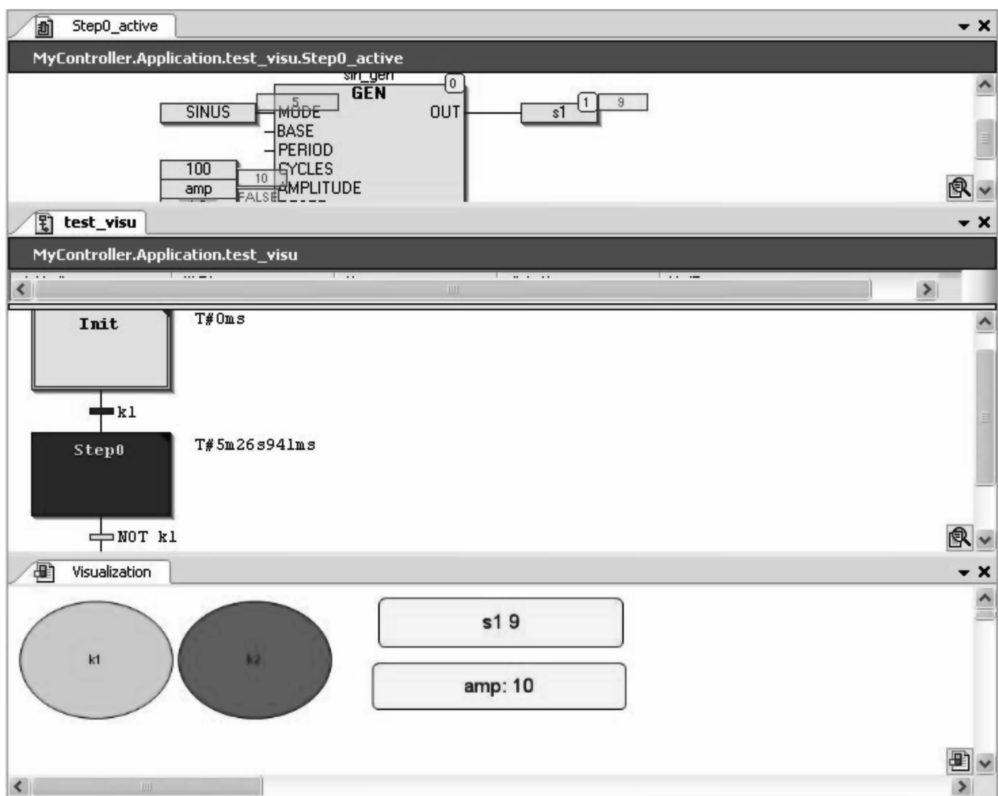


图 9-35　修改后的仿真运行

在运行中，用鼠标点击一下输入框，这时幅值就变为 100 了，如图 9-36 所示。

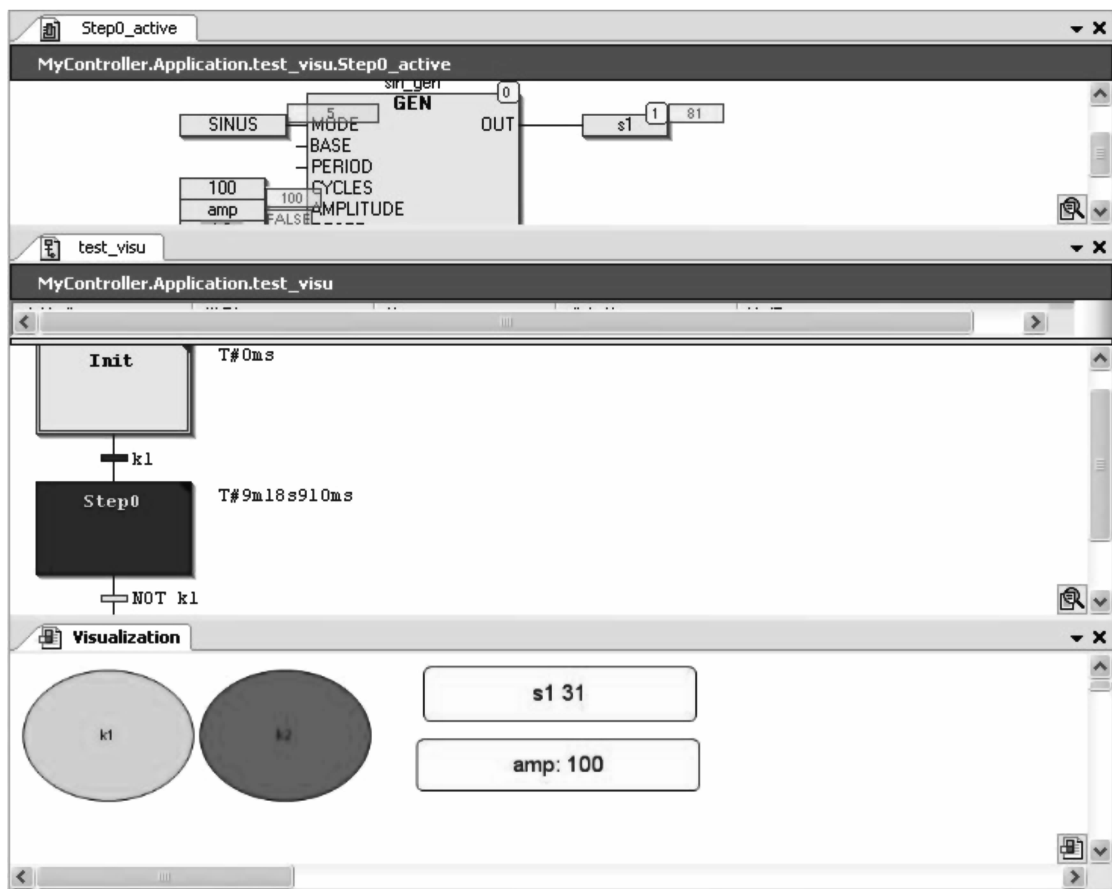


图 9-36　点击输入框 amp 变为 100

采用这种方法，你可以把一个个数值的输入变为一个个按钮，点击不同按钮，就赋不同的值，不用大量编程就能看到核心层的程序运行，从而为这个程序的稳定运行找到合适的方案。

3. 编辑一组运动控制界面

在 SoMachine 编程软件中，集成了众多的运动控制功能，为了快速使电动机动起来，完成各种运动控制功能，在视图界面中也定义好了符合 PLCopen 的视图界面。例如，图 9-37 所示，我们要使一个电动机轴 d1 使能，并做寻原点和定长运动。

在这里，我们不急于编写大量程序，只需把相应功能块放进程序编辑区，指定运动轴。然后我们将使用视图来操作这些功能块及在视图上填写某些参数。首先，我们建立一个视图，把鼠标箭头移到“Application”后，点击右键，打开菜单，选择添加对象，然后寻到视图，如图 9-38 所示。

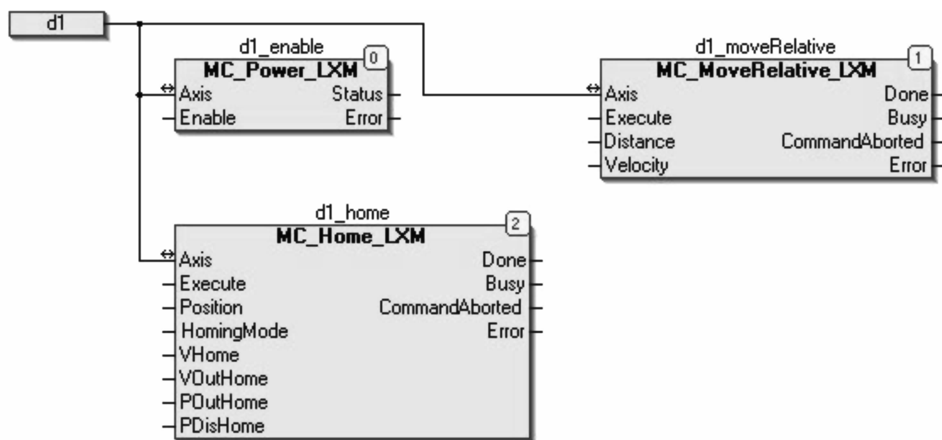


图 9-37　运动控制界面

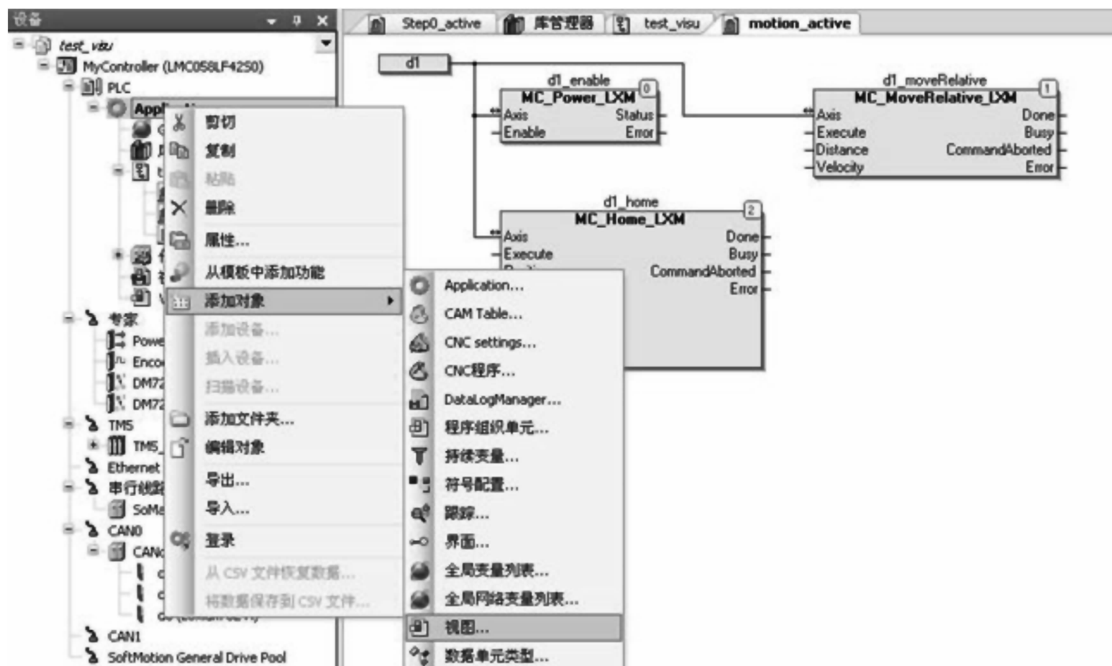


图 9-38　添加视图

点击视图，出现如下命名框，对添加的视图命名，例如 motion，点击“打开”，如图 9-39 所示。



图 9-39　命名视图

这样就建立了一个名字为 motion 的视图。在 motion 视图中，打开工具箱，如图 9-40 所示。

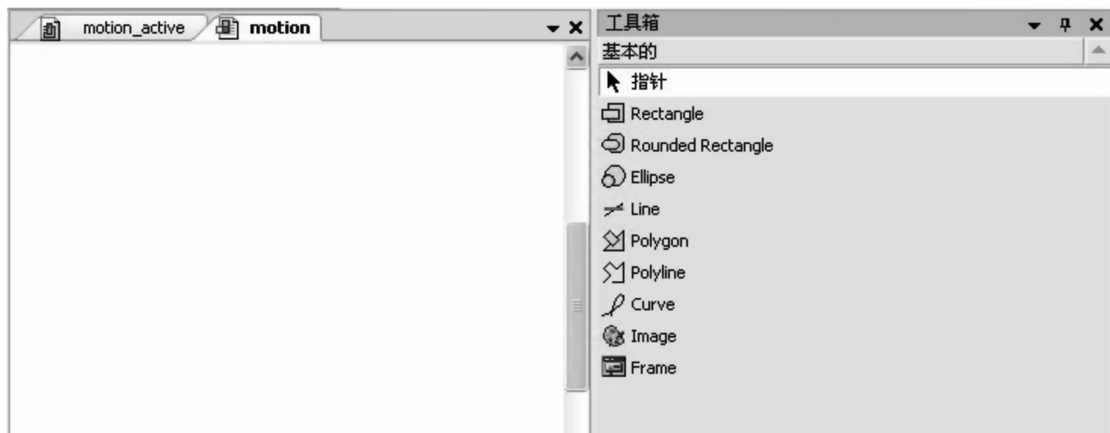


图 9-40　在视图中打开工具箱

点选工具箱的“Frame” Frame 模板，在视图编辑区划出一个边框，如图 9-41 所示。

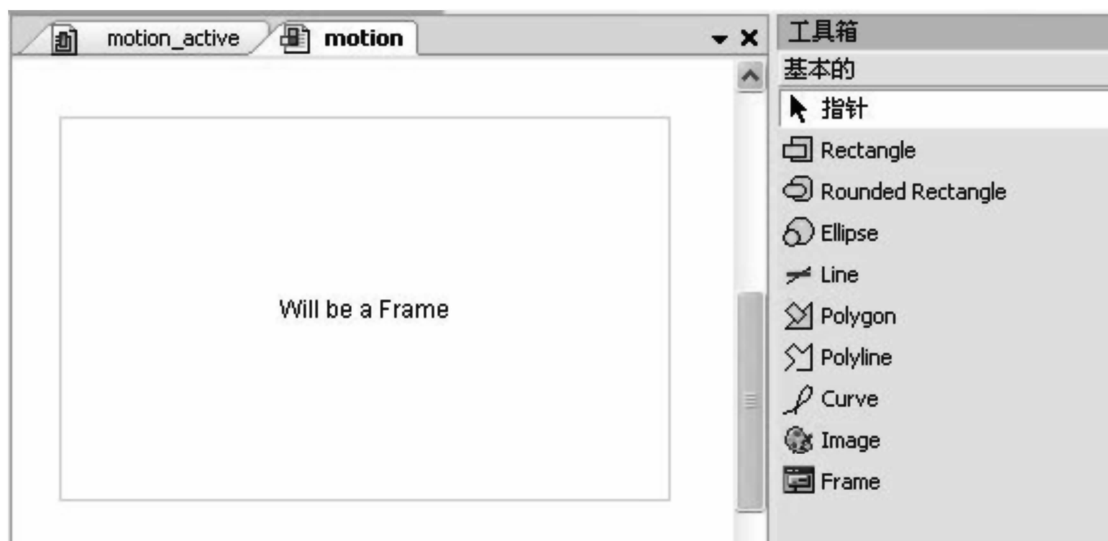


图 9-41　在视图编辑区划出边框

然后，在边框内点击鼠标右键，出现如图 9-42 所示菜单。

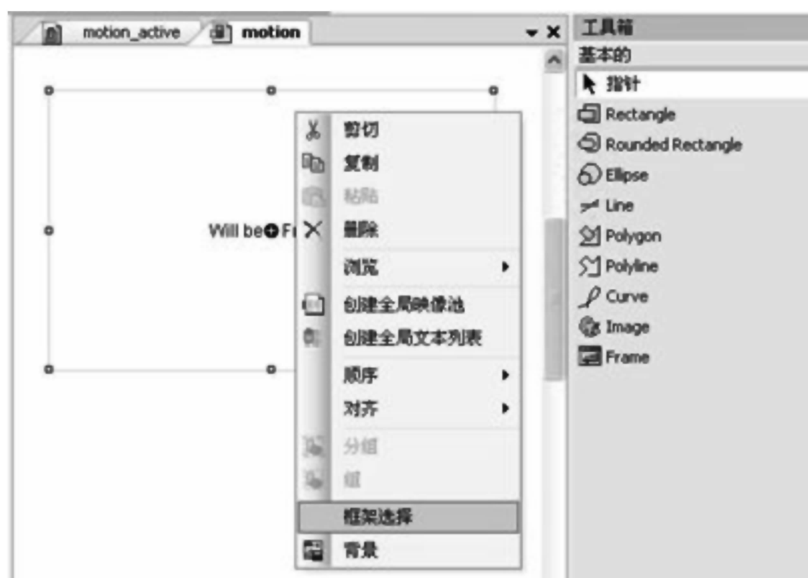


图 9-42　在下拉菜单中选择“框架选择”

在这个菜单中，选择“框架选择”。这样就出现了框架配置框，如图 9-43 所示，在框架配置的左边列出了很多视图模板。

在这个框架配置框中，移动上下的滑条，找出电动机轴使能的视图模板并调出，如图 9-44 所示。

按“确定”键，把所选视图调入编辑框内，如图 9-45 所示。



图 9-43　框架配置

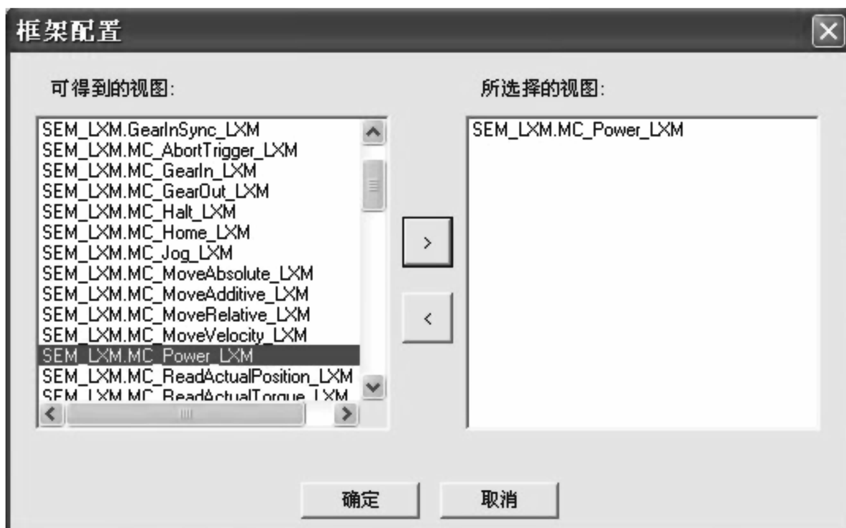


图 9-44　在框架配置框中选出相应功能

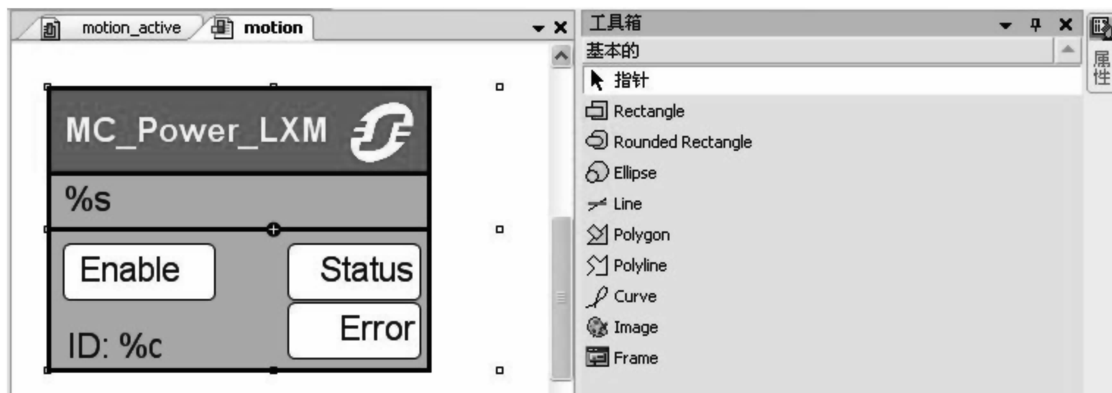


图 9-45　调入使能模板

点击右侧的“属性”，打开属性框，如图 9-46 所示。



图 9-46　打开模板属性框

在属性框里，找到参考输入，点击右侧的白框 ...，弹出输入助手，如图 9-47 所示。



图 9-47　打开输入助手

在输入助手框，找到在程序编辑区已放置的功能块 d1 _ enable，点击“确定”，完成对模板属性的组态。这样就使视图和功能块连接起来，在视图中就可以对此功能块进行操作。

完成模板组态后的属性，如图 9-48 所示。

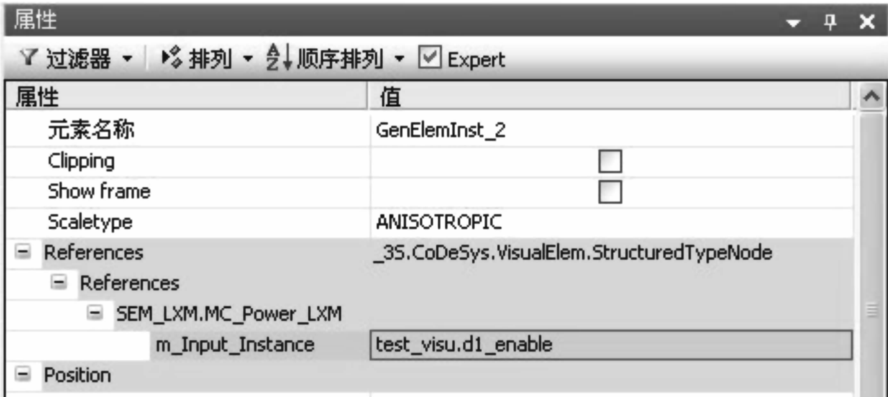


图 9-48　完成模板组态后的属性

同样步骤，我们找出寻原点和定长的视图模板，如图 9-49 所示。

对调入的视图模板的属性进行组态。对寻原点模板属性的组态，如图 9-50 所示。

对定长运动模板属性的组态，如图 9-51 所示。

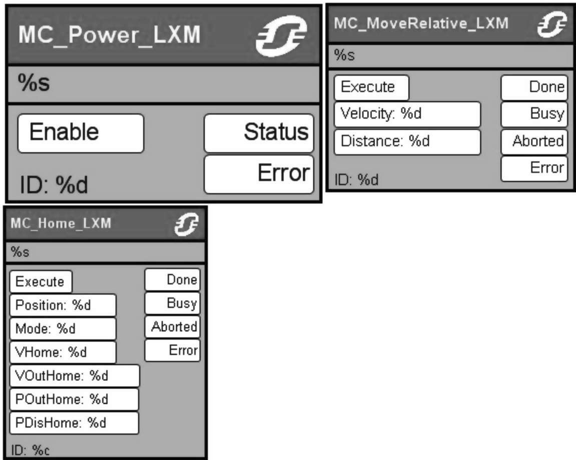


图 9-49　调入相应视图模板

编辑完成后，我们就可以通过视图对电动机轴进行使能、寻原点和走定长的操作了。由于在现场总线上所组态的轴是实轴，因此，这种结构的控制是不能做仿真实验的。如果要做电动机轴运动的仿真运行，就需要把运动轴设置为虚轴组。这在运动控制器 LMC058 上是可以做到的。因为运动控制器可以把运动轴设置为虚轴。如图 9-52 所示，把鼠标移到 SoftMotion 点右键，打开下拉菜单，选择添加设备。



图 9-50　寻原点模板属性的组态

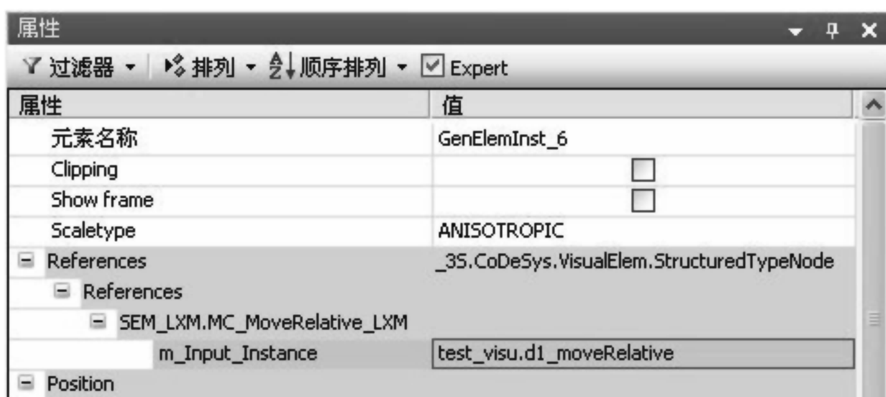


图 9-51　定长运动模板属性的组态

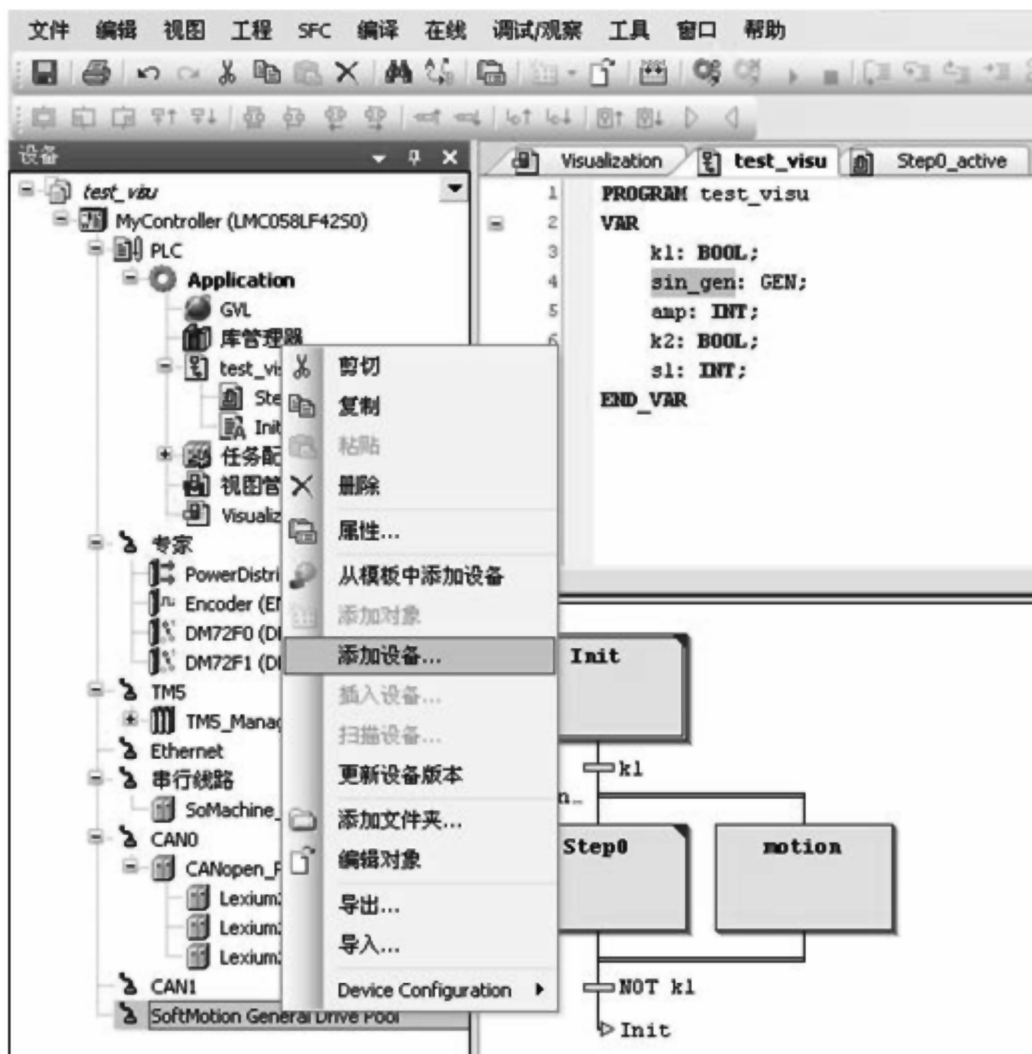


图 9-52　添加设备

在弹出的添加设备框中，添加虚拟驱动轴，如图 9-53 所示。



图 9-53　添加虚拟驱动轴

点击选中虚拟驱动下的虚轴后，点“添加设备”，即添加了一个虚轴 SM _ Drive _ Virtual。如果觉得这个名字过于繁琐，我们可以重新命名这个虚轴。如图 9-54 所示，在默认名下打开菜单，选择“属性”。在属性栏目下，修改设备名。默认设备名如图 9-55 所示。

修改名字为 d1，如图 9-56 所示。

按“确定”后，在设备栏，我们看到建立的虚轴 d1，如图 9-57 所示。

这样我们就可以用功能块视图对虚轴 d1 进行操作。这样的好处是：我们可以在没有实轴的条件下，只通过运动控制器就可以仿真多个轴的运动，包括多个轴的同步运动，例如：电子凸轮、CNC 等。如图 9-58 所示，我们双击 motion 框，建立一个 CFC 轴运动程序。

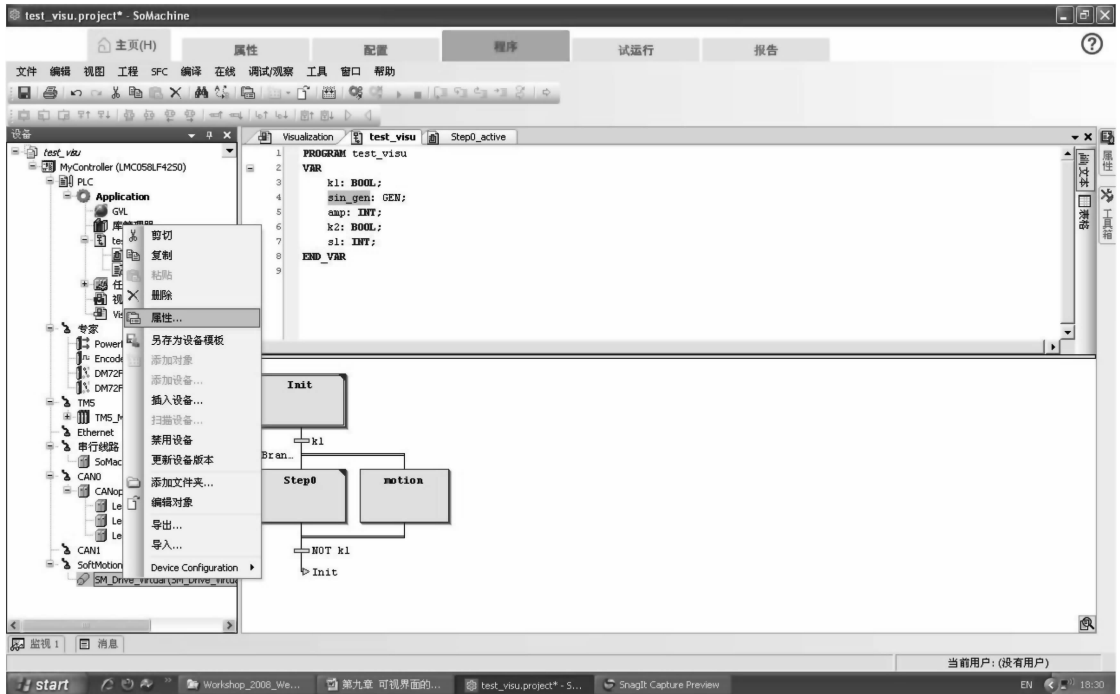


图 9-54　在本条目下打开菜单



图 9-55　默认设备名



图 9-56　修改设备名

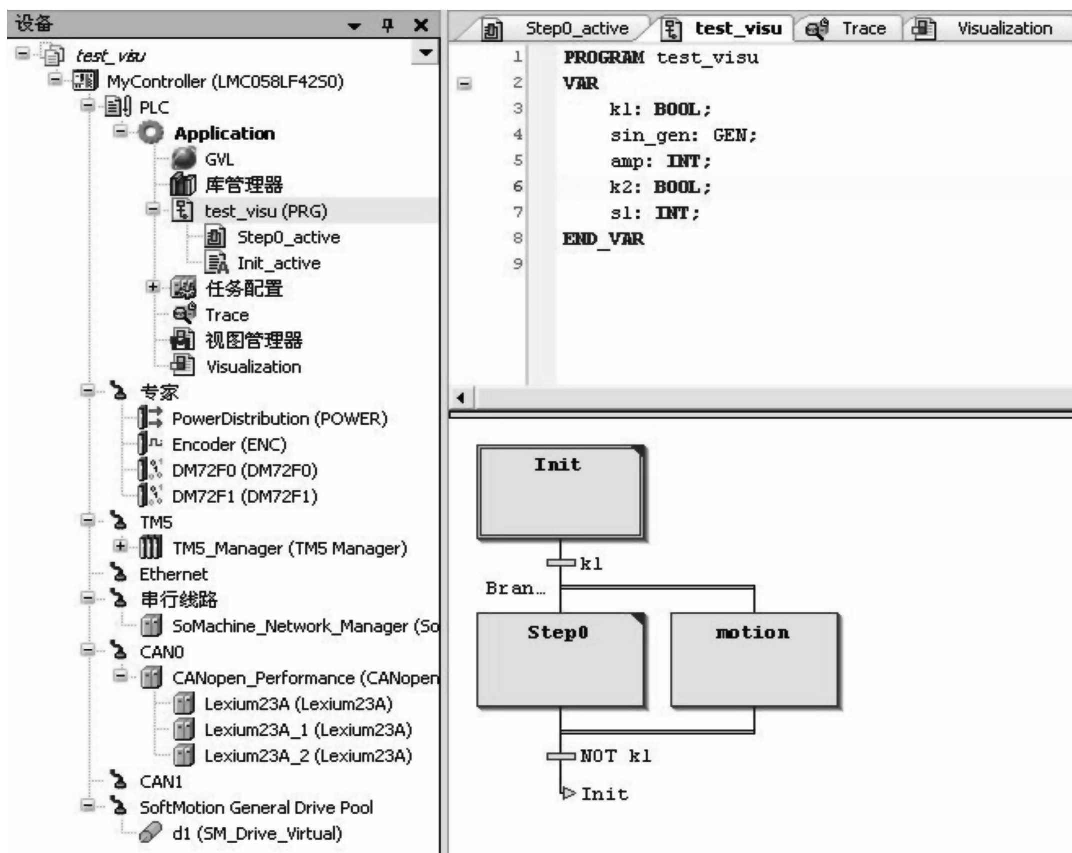


图 9-57　虚轴设备名



图 9-58　建立一个轴运动程序

在程序编辑区，调入使能功能块、速度功能块、位置功能块，如图 9-59 所示。

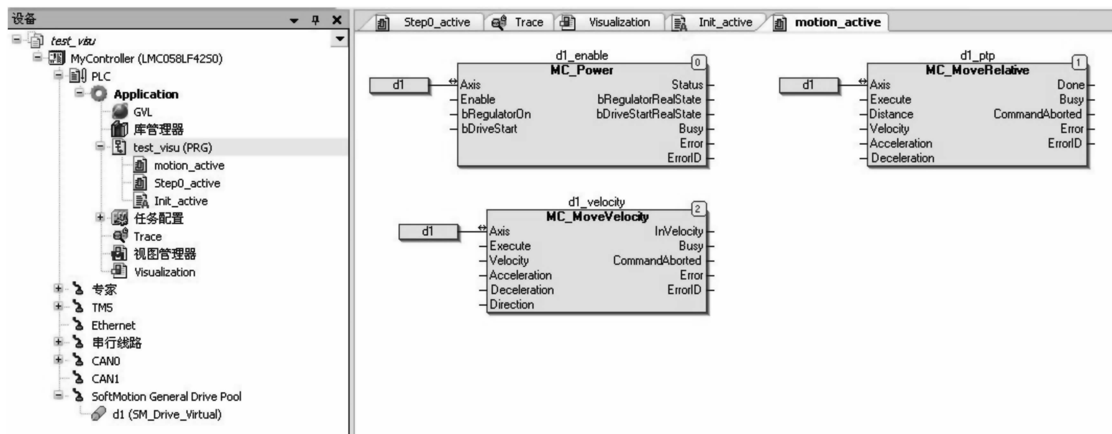


图 9-59　编辑运动控制程序

根据这个程序，我们建立一个 motion 视图操作。如图 9-60 所示，把鼠标移到“Application”点右键，打开菜单，添加视图。

创建 motion 视图，如图 9-61 所示。

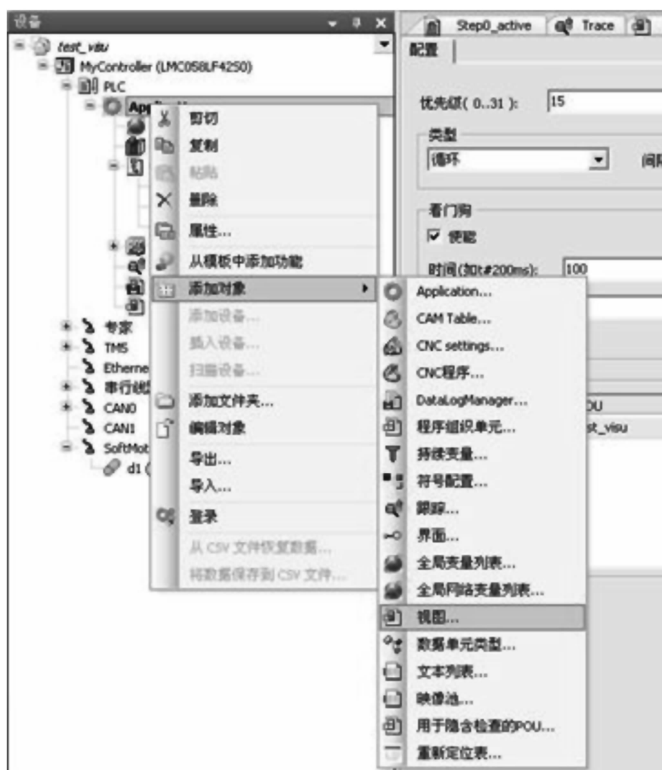


图 9-60　添加视图



图 9-61　创建 motion 视图

在视图界面，打开工具箱，点选框架 Frame ，在编辑区划出一个框架范围，如图 9-62 所示。

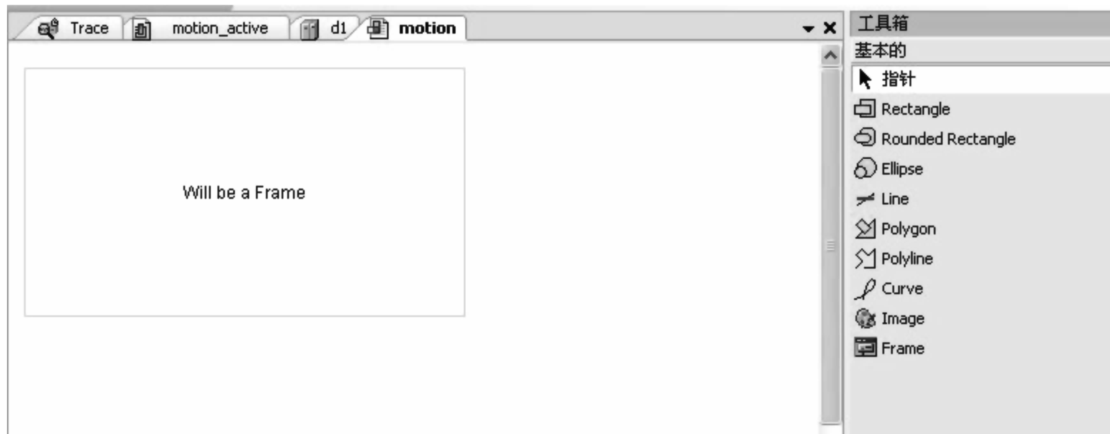


图 9-62　编辑视图控制

在框架中，点右键打开菜单，选择“框架选择”。在弹出的框架配置中，选择使能模板，如图 9-63 所示。



图 9-63　在配置框中选择相应模板

确定后，模板被调入视图，如图 9-64 所示。

选中调入的模板后，打开属性框。在属性框中的参考 References 栏目下，点击右侧的输入助手，在输入助手框找到已经编辑好的使能功能框 d1 _ enable，如图 9-65 所示。

同理，我们在视图编辑区调入速度模板、位置模板并编辑好对应属性，motion 视图如图 9-66 所示。

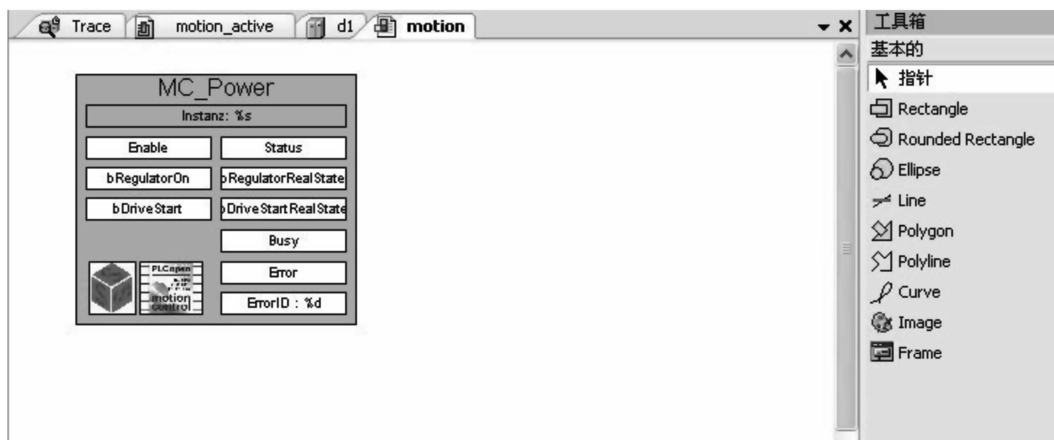


图 9-64　调入使能模板

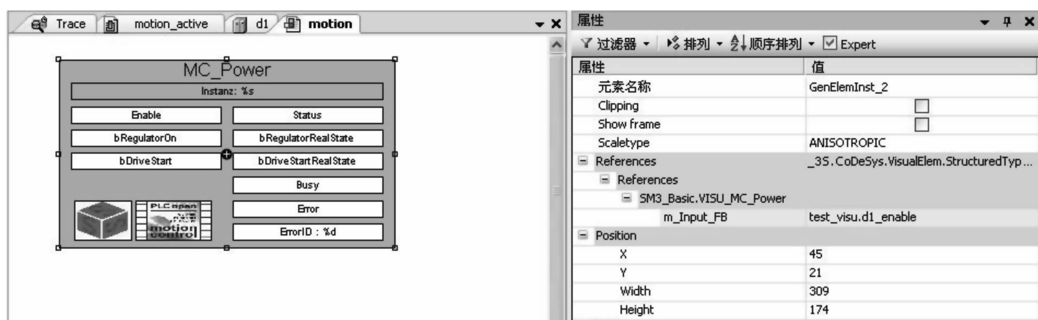


图 9-65　模板属性编辑

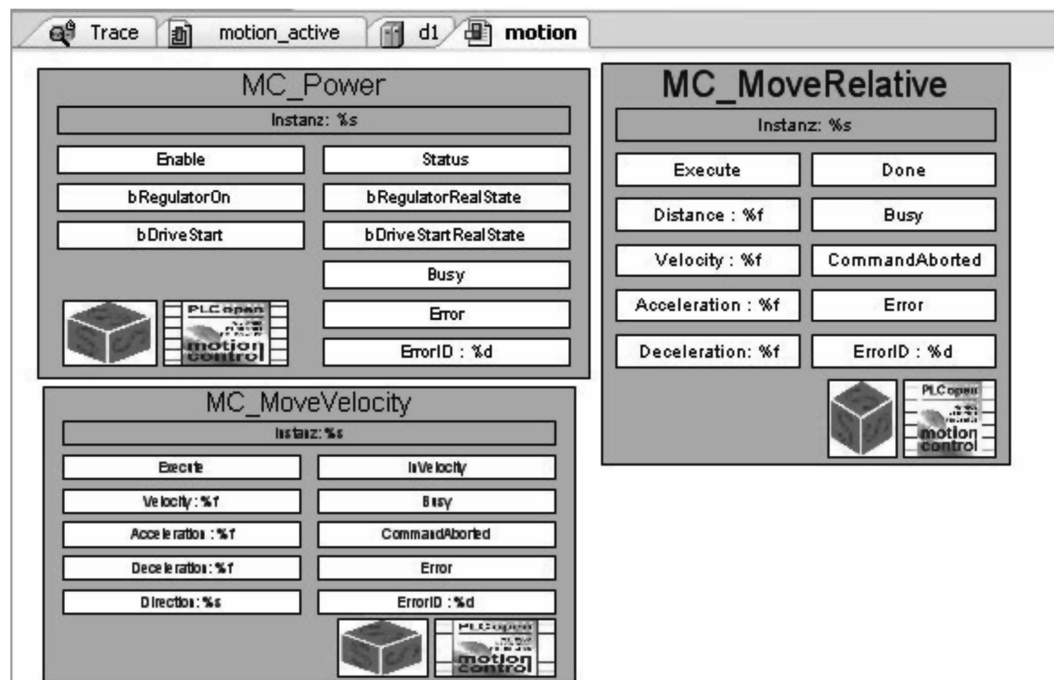


图 9-66　motion 视图

为了直观显示轴的运动，我们可以再调入一个轴运动的模板，即在打开的框架配置中，选择轴的圆周运动，如图 9-67 所示。

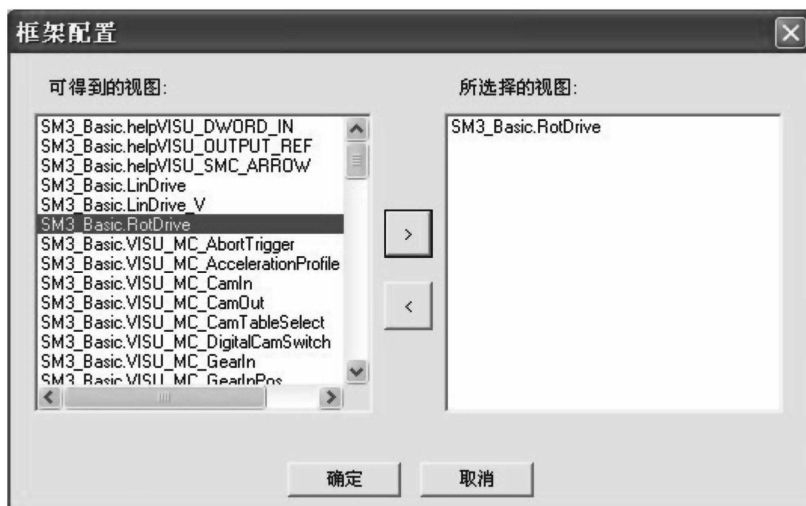


图 9-67　选择轴的圆周运动

确定后，在视图编辑区会看到一个圆形指针，如图 9-68 所示。

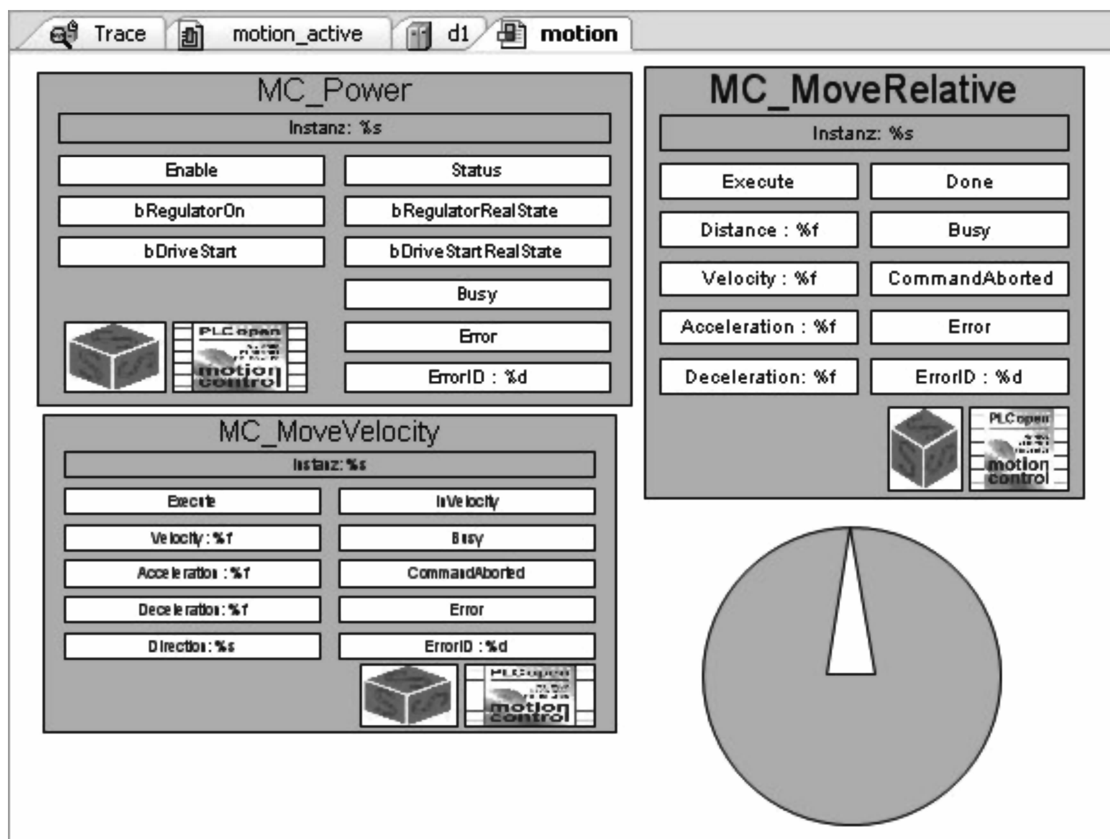


图 9-68　轴动作指示

选中这个轴模板，打开“属性”，属性的参考值为轴的名字 d1，如图 9-69 所示。

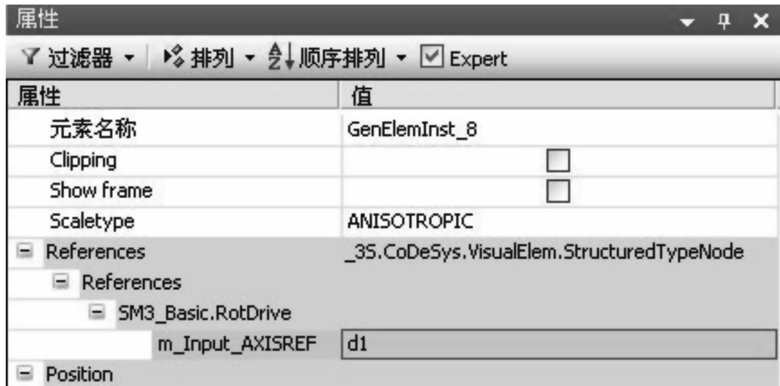


图 9-69　轴的属性

视图做好后，我们就可以连接上 LMC058 进行轴运动的仿真了。如图 9-70 所示，在视图中我们可以使能轴并且操作速度运行。也可以操作轴做定位控制。如图 9-71 所示，一旦轴位置到达，则 DONE 信号变绿。

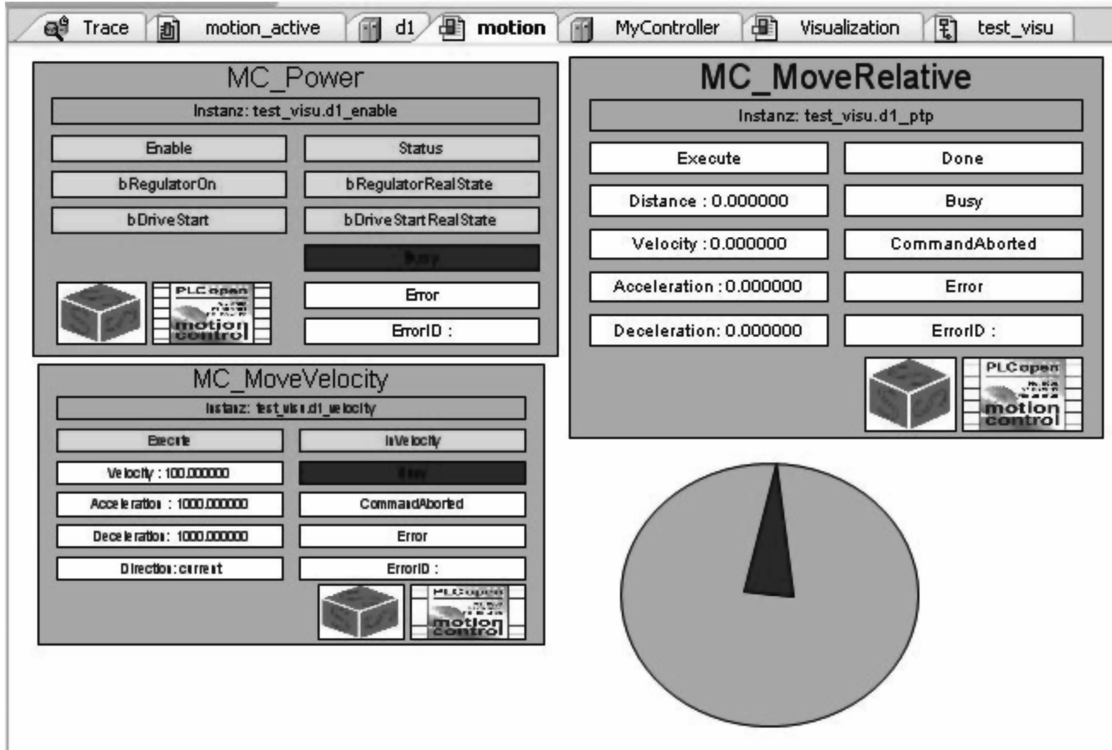


图 9-70　速度控制

4. 编辑一个跟踪曲线的动态显示

要编辑一个跟踪曲线的动态显示，首先就要建立并组态好一个曲线跟踪记录器即跟踪。把鼠标移到应用“Application”点鼠标右键，打开菜单，如图 9-72 所示。

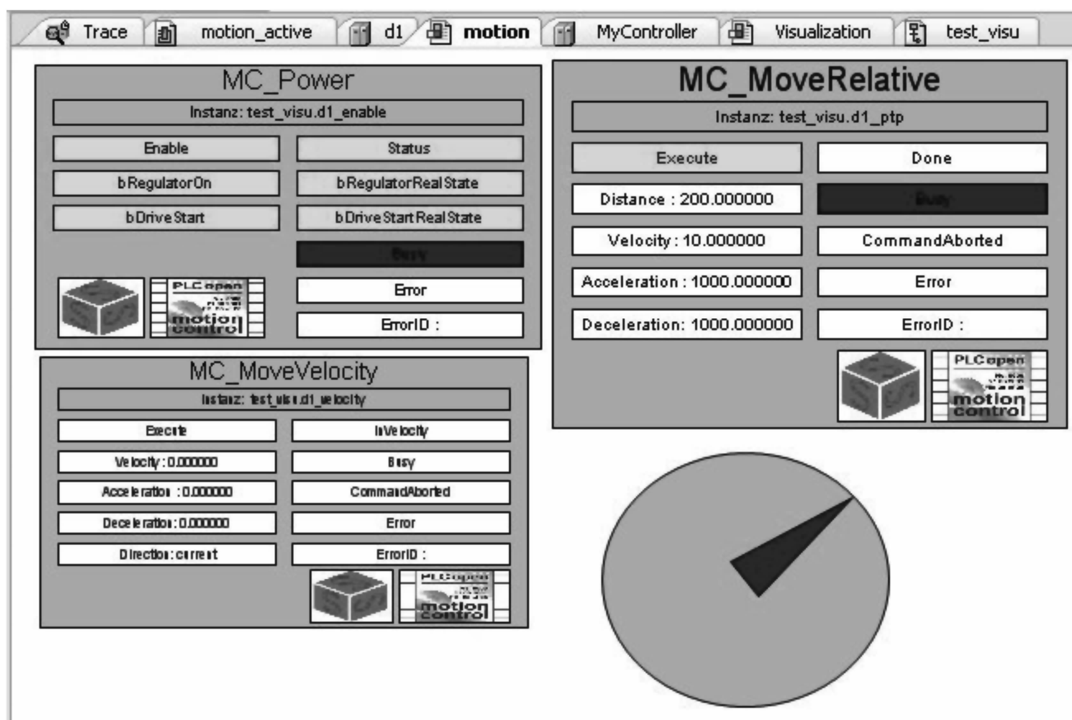


图 9-71　位置控制



图 9-72　建立一个跟踪

在打开的菜单中，选择跟踪。在弹出的框中命名此跟踪器的名字，如 Trace，如图 9-73 所示。



图 9-73　命名跟踪器

按“打开”，得到一个跟踪器框架，如图 9-74 所示。

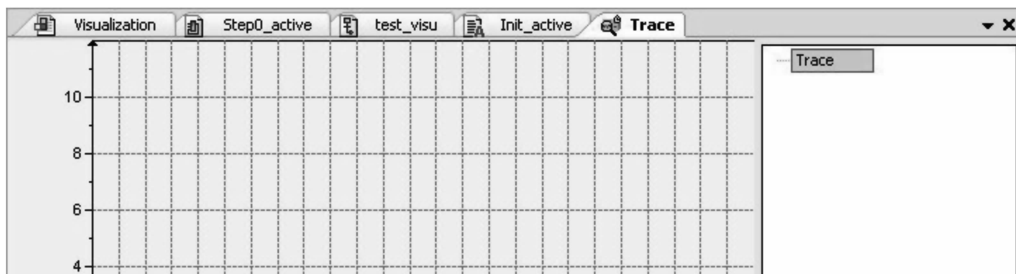


图 9-74　建立一个跟踪器

我们对这个跟踪器框架进行配置，点右键，打开下拉菜单，如图 9-75 所示。



图 9-75　打开 Trace 菜单

把鼠标移到 Trace 点右键，打开菜单，选择配置，打开如图 9-76 所示的配置界面。

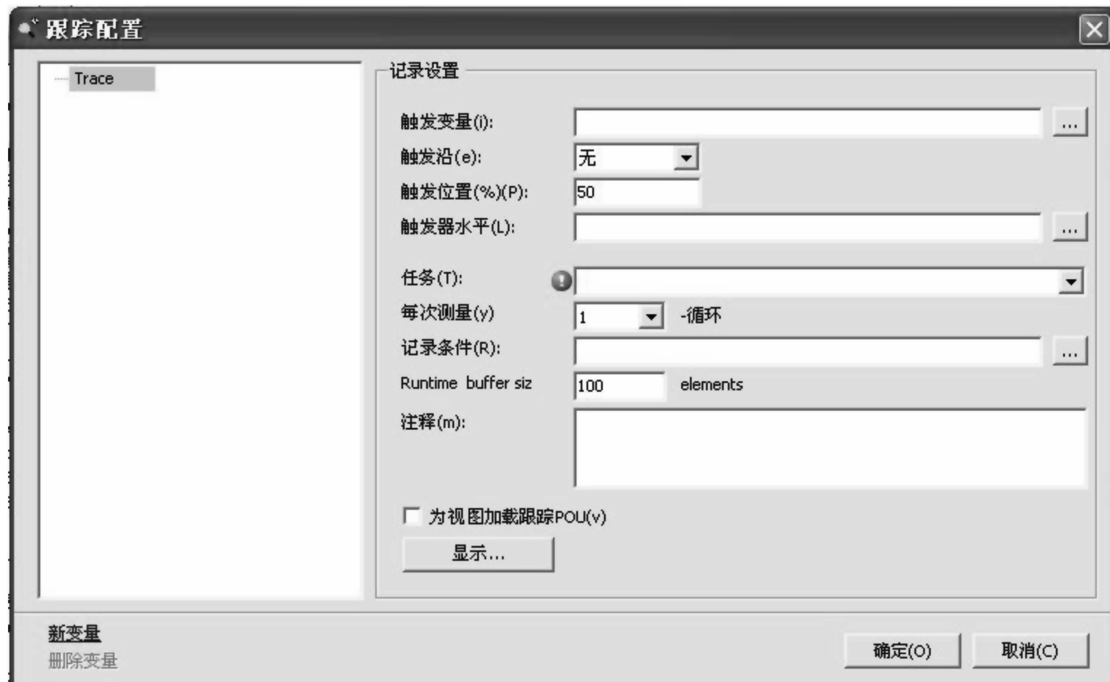


图 9-76　跟踪记录器配置界面

配置一个曲线跟踪器如图 9-77 所示。



图 9-77　配置一个曲线跟踪器

在这个跟踪器里，要记录的曲线是 s1。要注意的是：我们要把这个曲线变化 s1 加载到视图，所以一定要激活为视图加载跟踪，如图 9-77 所示，在为视图加载跟踪前打勾。

点击“确定”后，在设备栏目下，有 Trace 名出现，如图 9-78 所示。



图 9-78　添加了一个跟踪器 Trace

打开已编辑好的视图 Visualization，在视图下，打开工具箱，如图 9-79 所示。



图 9-79　打开视图中的工具箱

点击工具箱中的复杂控制 Complex Controls ，打开此控制下菜单，如图 9-80 所示。

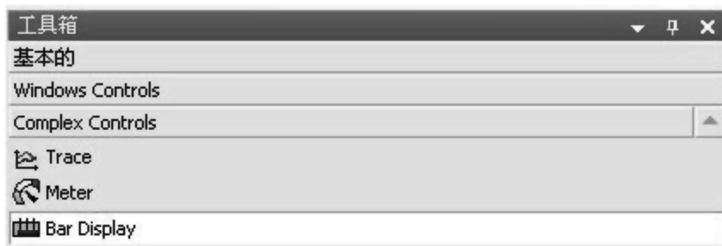


图 9-80　复杂控制下菜单

选择 Trace 条目，在视图编辑区拉开一个显示框，如图 9-81 所示。

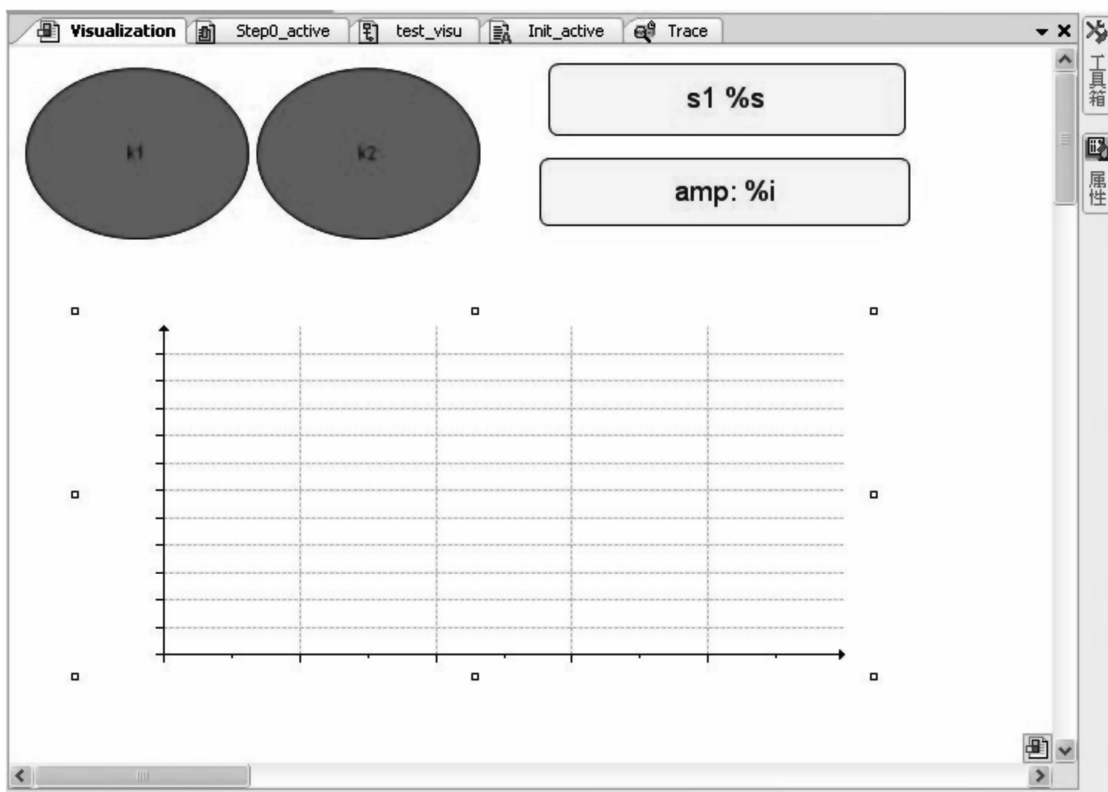


图 9-81　在视图编辑区拉开一个显示框

点击“属性”，打开属性框，如图 9-82 所示。

在 Trace 栏，点击右侧帮助。

打开输入助手，在这里，选择要显示的曲线记录器 Trace，如图 9-83 所示。

完成后的配置如图 9-84 所示。

配置完成后，我们就可以采用仿真来看一下效果。仿真并运行程序，这时在视图的显示框中就显示出了动态的曲线 s1。图 9-85 所示是曲线振幅为 10 的情况。



图 9-82　显示框属性



图 9-83　选择要显示的曲线记录器



图 9-84　视图中动态显示图的配置

当幅值改变为 100 时的仿真显示，如图 9-86 所示。

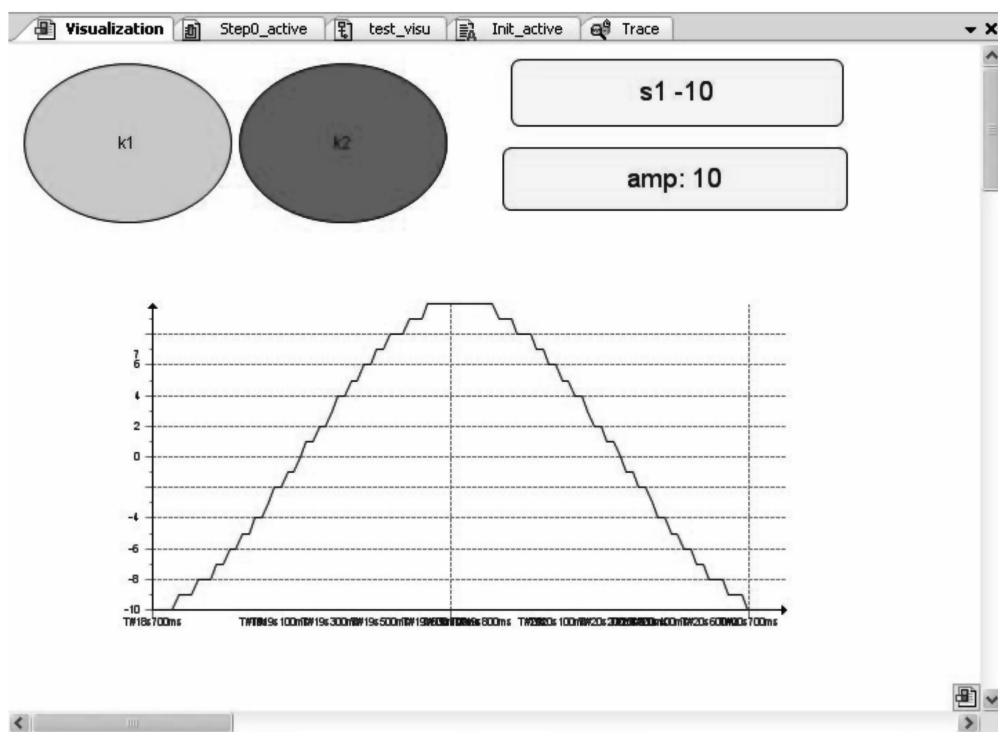


图 9-85　曲线的动态记录显示

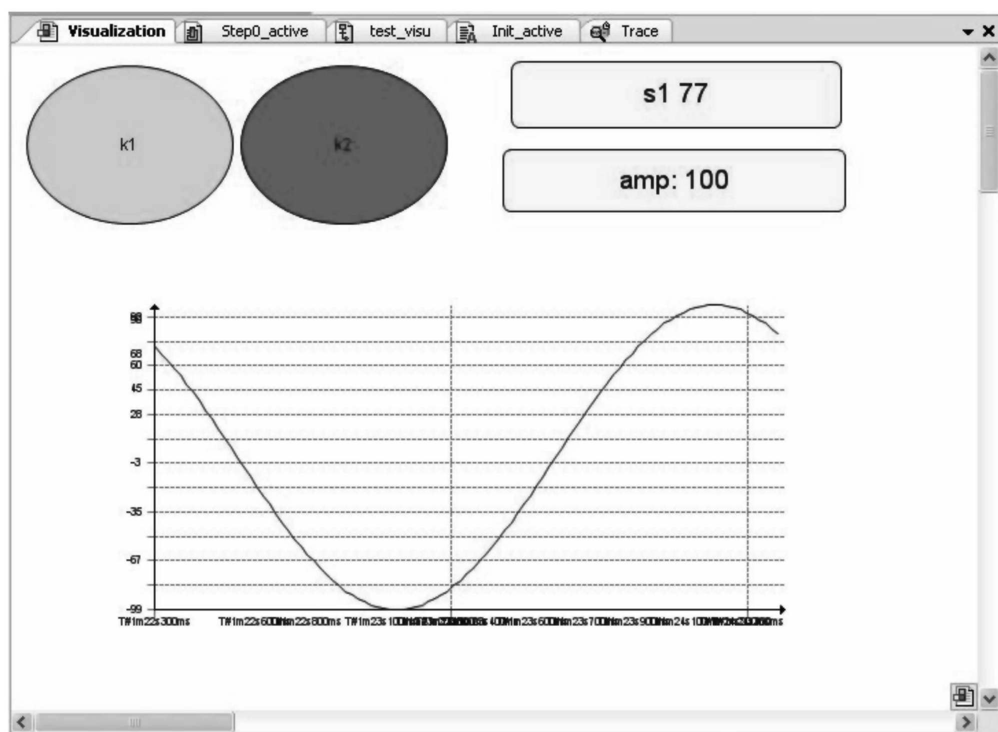


图 9-86　幅值改变为 100 时的仿真显示

9.3　复用可视界面

如图 9-87 所示，从设备栏中，我们看到已经有两个视图操作界面，一个是 motion，另一个是 Visualization。通常情况下，在整体调试时，我们希望把各个界面能集中到一个界面里，或在一个界面中，也可以操作另一个界面。

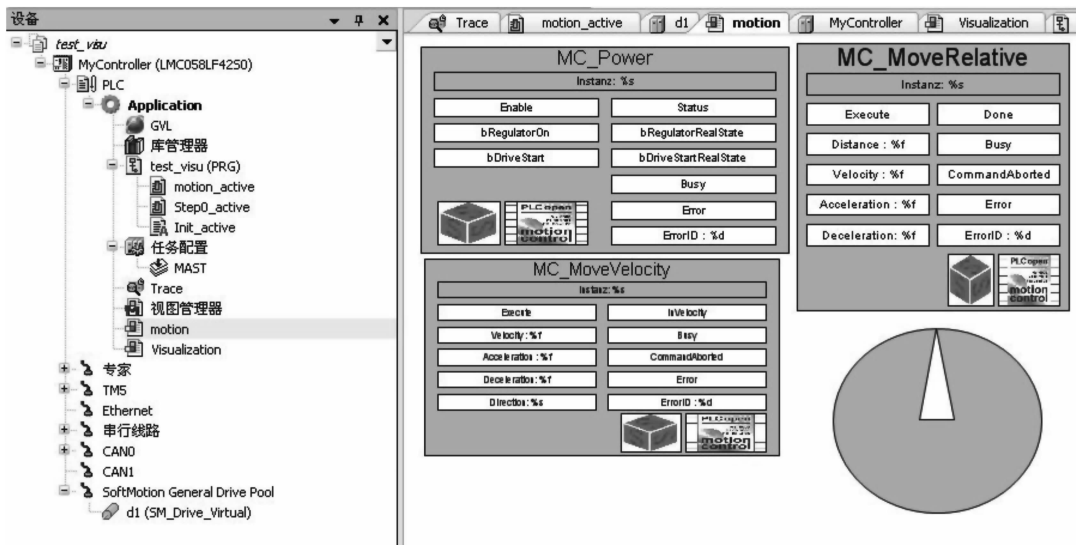


图 9-87　视图操作界面

这种编辑好的界面，都可以作为模板供其他界面调用。这就是可视界面的复用功能。它的操作类似于对专用模板的调用，只不过模板名为自己创立的界面名字。例如，在 motion 视图界面下，我们打开工具箱，使用框架 Frame 在界面画出模板区，在模板区中，打开菜单选择结构框架，在弹出的框架配置栏，选择已有的视图 Visualization，如图 9-88 所示。



图 9-88　选择已编辑的视图

点击“确定”后，已有视图调入本编辑视图中，如图 9-89 所示。

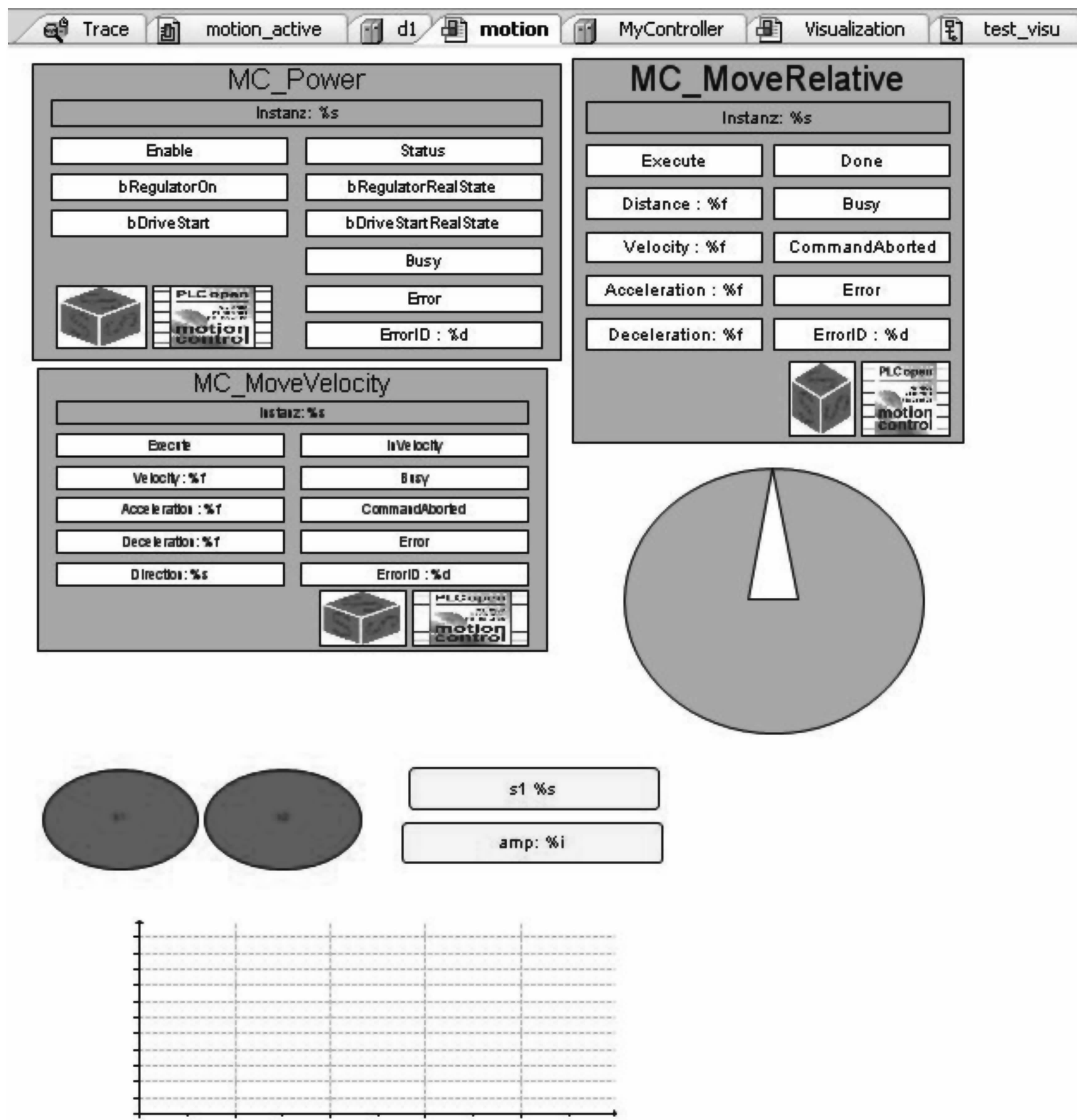


图 9-89　调入一个操作视图

这样，我们就可以在一个界面中操作程序了，在这个界面的仿真如图 9-90 所示，刚开始运行时，程序在初始化框中扫描，虽然对轴施加了使能信号，但从状态看，运动轴并没有使能，这是因为 motion 框中的程序并未被扫描启动。我们在操作界面点击 k1，打开程序流，使运动控制程序得到扫描，因此运动轴使能，如图 9-91 所示。

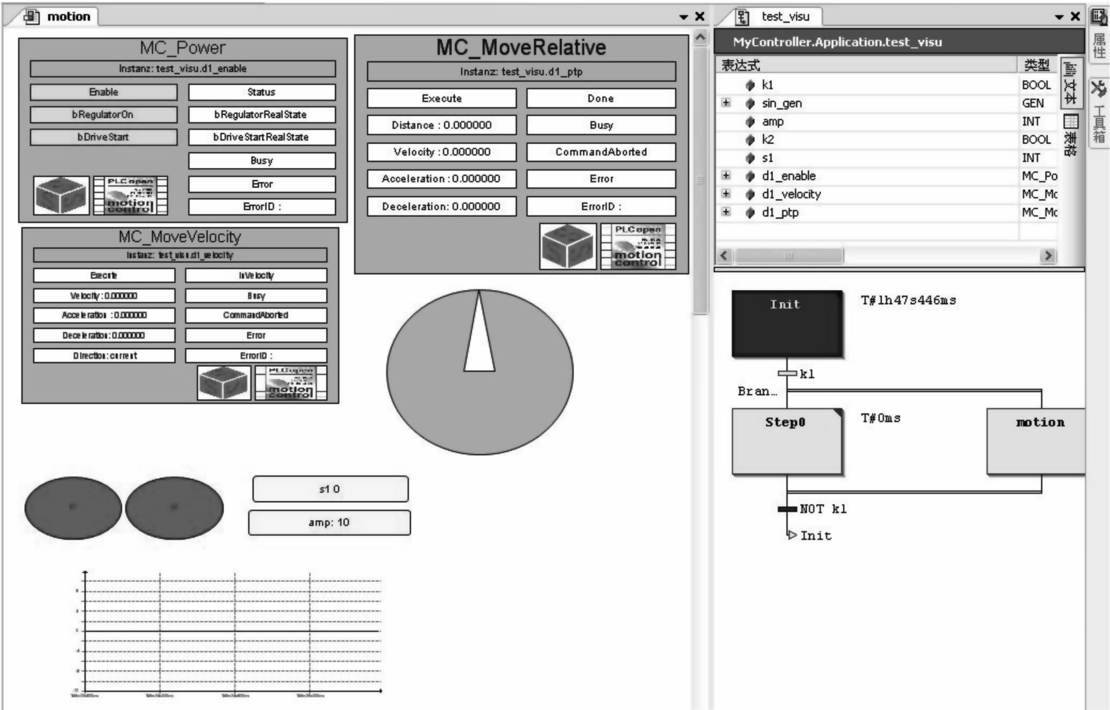


图 9-90　初始化阶段

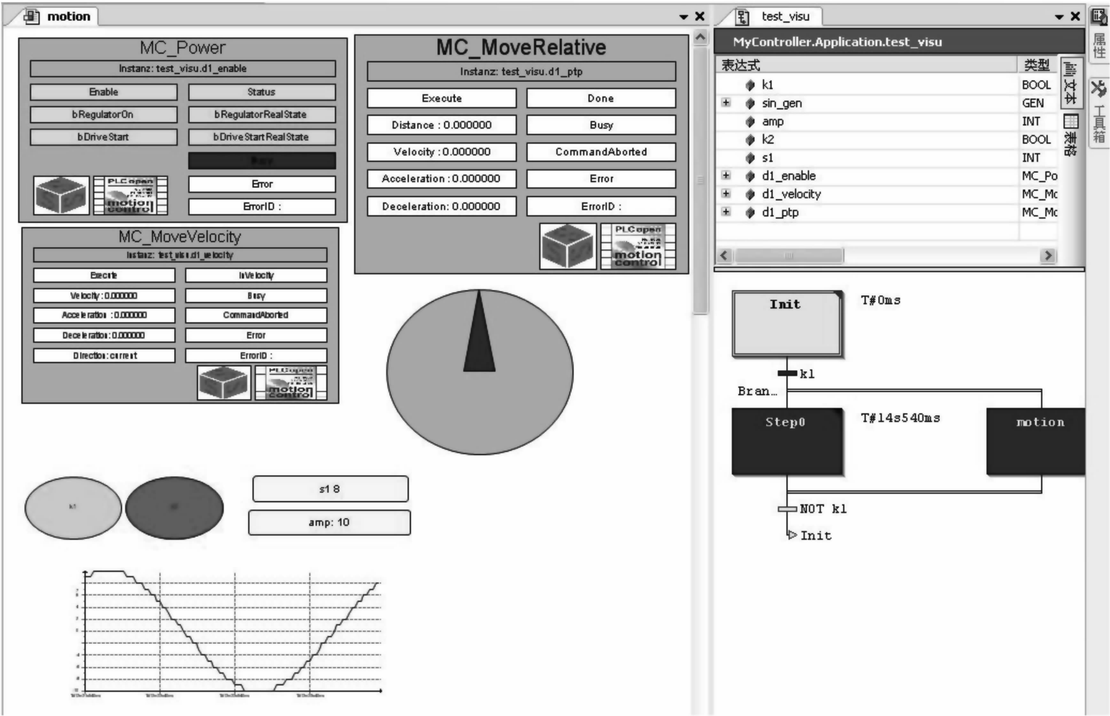


图 9-91　运行阶段

第 10 章　应用库文件

在编写 SoMachine 程序时，工程师们常常遇到一些功能块能够找到并可调入编辑区，但打开后发现是一个空框，没有引脚，或引脚都没有定义的情况。有的时候更是连功能块也找不到。还有的情况是，把一个已经应用过的程序再用其他计算机调用就会出现很多问题，究其原因都是库文件惹的祸。那么什么是库文件？它是怎样支持着我们的程序？它的建立和应用又能给编程带来哪些益处？本章会详细阐述。

10.1　应用库文件概述

我们在进行硬件的配置和软件配置后，还需要编制应用程序。所有这些操作做完后就要进行编译了，编译完后，我们会发现出了很多错误，这使初学者往往不知所措。而这编译的背后支持就是各种库文件。它们有系统文件，支持着查找并纠正各种硬件配置的错误和软件语法问题；标准库文件，支持着标准功能的实现；还有各种专有的库文件，支持着诸如包装、起重、物料输送、数控及机器人等专有功能。那么什么是库文件呢？库文件是：

1） 由 CoDeSys 软件提供的，可重复使用的公共功能块集群，或由其他软件供应商提供的某一领域专有的功能块集群或由你自己开发的适合本领域重复使用的功能块集群。

2） 把这些功能块集群放在一个库里，就构成库文件。

3） 库的管理员把不同应用的功能进行分类，从而方便用户查找使用不同类型的应用功能块集群。

4） 非标准库可以根据需要通过库管理员加入到应用库，以支持专门的应用场合。

根据库文件支持的功能不同，可以把库文件分为系统库文件、应用库文件以及专门库文件。先让我们来看看系统库文件。

1. 系统库文件

系统库文件顾名思义是一个支持软件系统的文件，它包括对软件结构和语法编写的支持以及标准 I/O，即输入/输出的支持。通常情况下，这个文件库会在开机后自动调入到控制器中。类似于计算机的操作系统。所以一般不用手动添加。

2. 应用库文件：即用户库文件

它是支持一个基本应用的文件库。

Util：它包含了各种数学运算功能，对位和字节的操作等功能。

Standard：包含了计时器（Timer）功能，计数器（Counter）功能等。

这些功能都是一台 PLC 和控制器必须具备的，因此在应用中都会被默认调入控制器。还有一些是按照用户需求来导入的应用库，如：Toolbox、PLCopen 等，这些库文件都是根据控制的设备配置的通用标准功能块。例如：运动控制的各种功能块。这些功能块的组织结构如图 10-1 所示。

3. 专门库文件：由专业厂商提供的库文件

它是根据硬件产品的环境而配置的应用库。例如：为施耐德电气的 PLC 控制器配置的

M238PLC 库，以及为这个产品而配置的高速计数和为 LMC058 运动控制器配置的运动控制库 PLCopen。SM3 _ BASIC 运动功能库如图 10-2 所示。

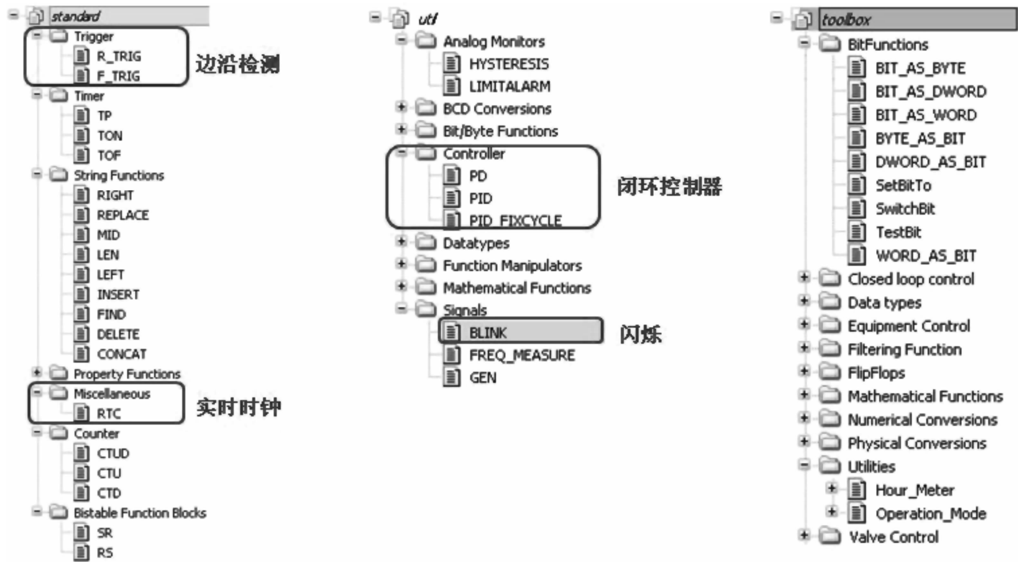


图 10-1　应用功能块的组织结构

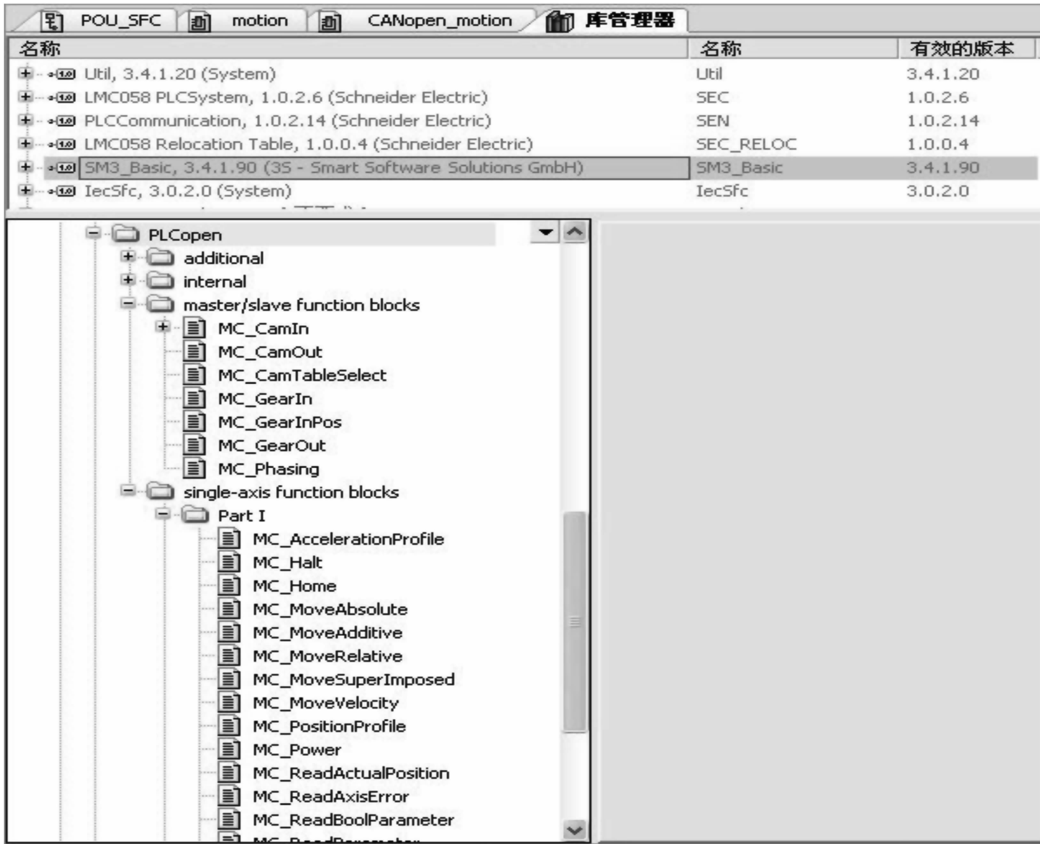


图 10-2　SM3 _ BASIC 运动功能库

为第三方产品配置的应用库，如 OSCAT 应用库（开放资源），如图 10-3 所示。



图 10-3　第三方产品应用库

当应用时，可以把相应库文件调入。对库文件的调入、更新和删除参见下节库文件的管理。

10.2　库文件管理

在编程应用时，对安装在计算机上的库和供应商提供的库文件进行调入。可以按如下步骤进行。如图 10-4 所示，在程序编辑界面，点击工具栏，并且在工具栏下拉菜单中选择库。



图 10-4　对功能库的调入

打开库后，会出现如图 10-5 所示界面。在这个界面中，会看到已安装的库和提供这些库文件的供应商。并且可以按照分类查看文件库名称。在这些库的分类中，我们列出了应用类、通信类、控制器类、设备类、内部资源类、解决方案类、系统类和应用工具类。在实际编程中，如果调入到程序编辑区的功能块没有定义，那么就可以查看相关库是否存在以及版本是否合适。如果库文件不存在或版本过期，就可以启动安装来添加新库或更新库文件版本。

点击如图 10-5 所示界面的“安装”，进入选择库界面。在选择库界面，可以通过查找范围栏，查找库文件在计算机的位置或 U 盘复制的库文件，如图 10-6 所示。选择库文件后，按“打开”键，即把库文件调入系统中，如图 10-7 所示。在已安装的库文件中，就可以看

到新加入的库文件了。

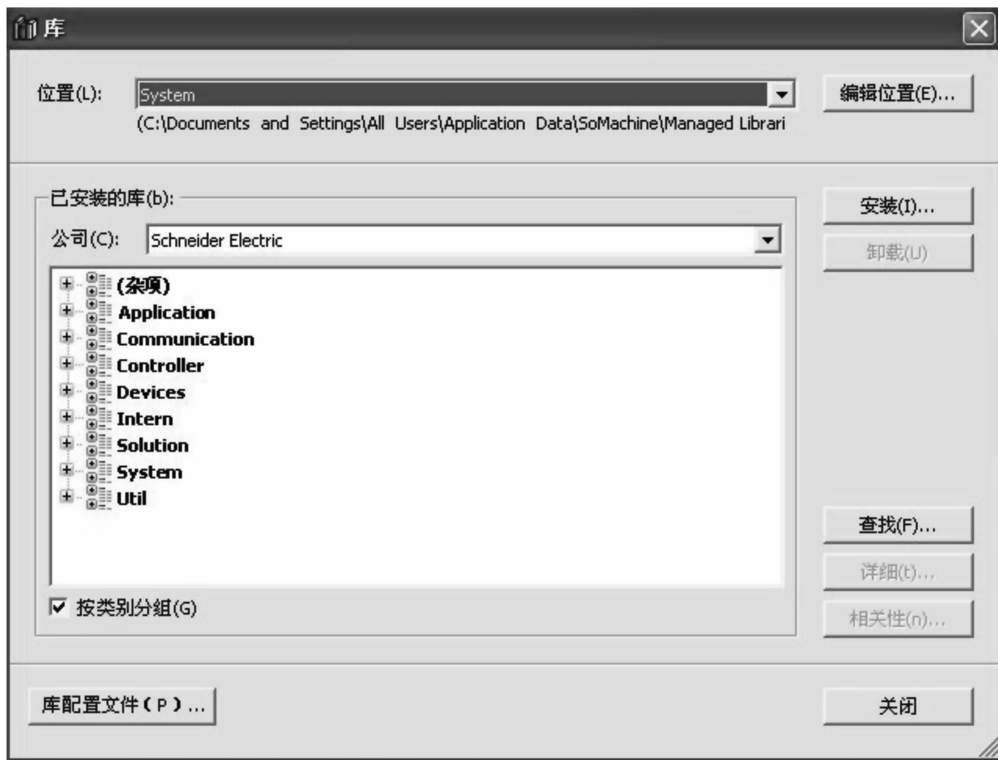


图 10-5　配置功能库

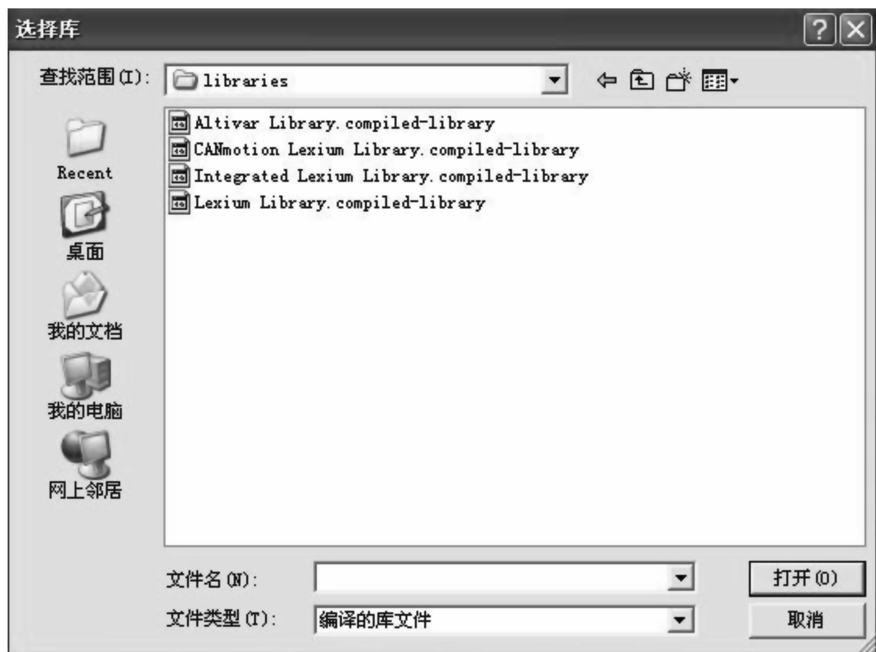


图 10-6　计算机中的库文件

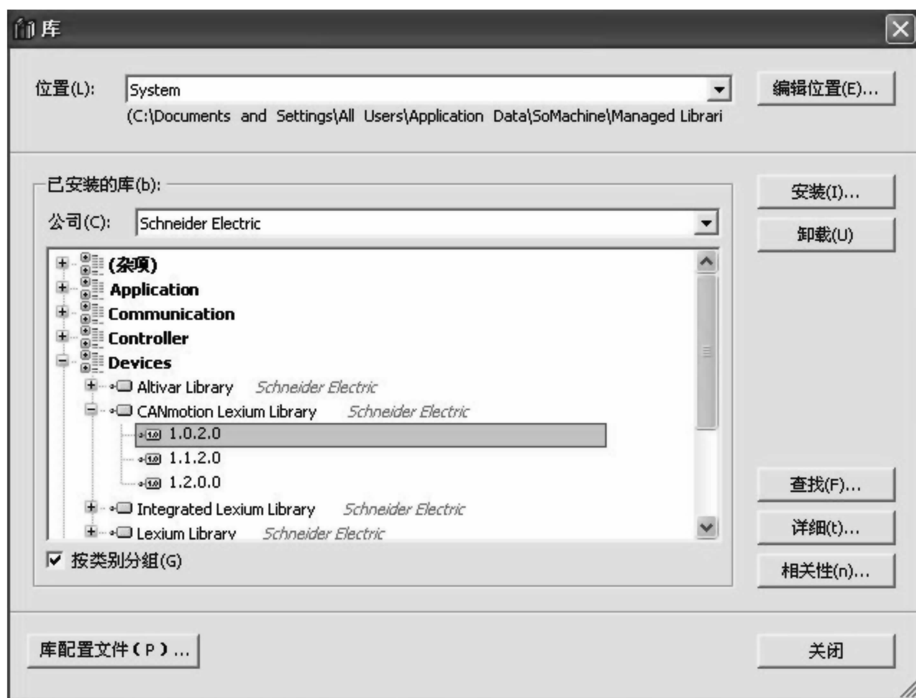


图 10-7　库文件调入

也可以通过如图 10-8 所示的库管理器来添加库文件。点击图中所示的库管理器，按照右边出现的导引来安装即可。



图 10-8　库管理器

安装完后，可以通过点击安装好的库文件来查看库内的功能块及其说明，如图 10-9 所示。第一步点击库管理器。第二步添加库文件。第三步点击已安装的库文件，打开此库文件下的功能块。

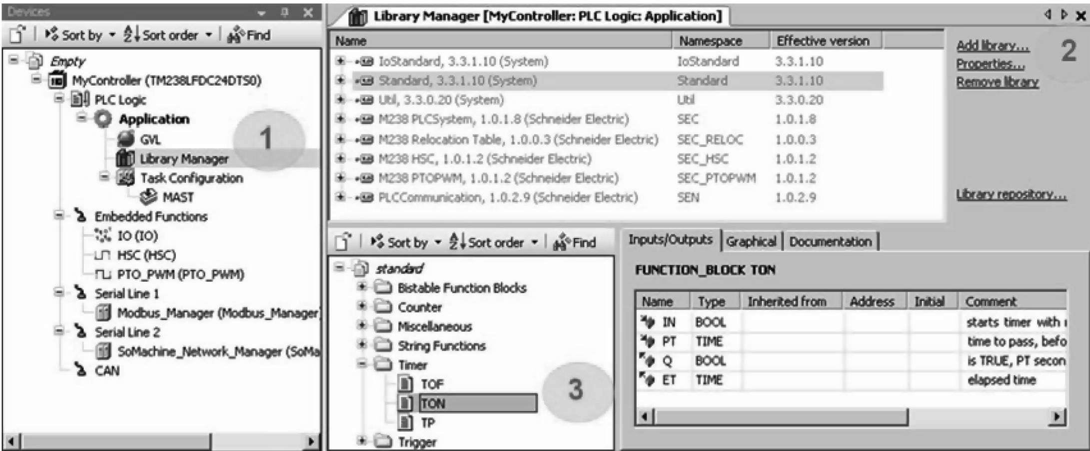


图 10-9　使用库管理器安装查看功能块

10.3　库文件的建立

除了软件厂家提供的功能库外，我们也可以根据自己的应用经验，把某些专用功能块的集群整理出来，建立自己的专用功能库。例如，在一个灌溉的程序中我们编辑了很多这个行业的功能块，如图 10-10 所示。

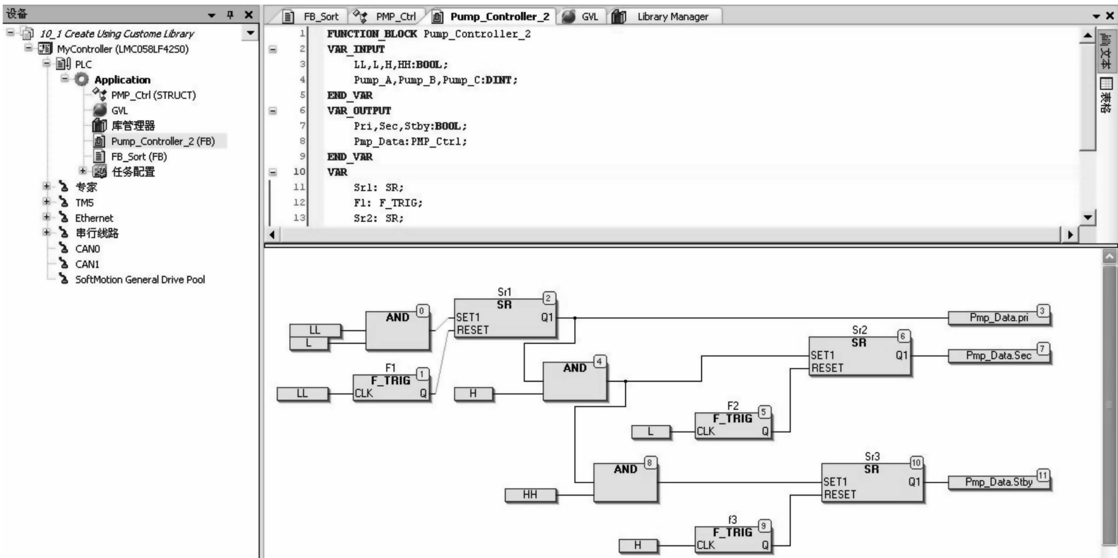


图 10-10　编辑的用户功能块

选中功能块，点击鼠标右键打开下拉菜单，选择“导出”，如图 10-11 所示。

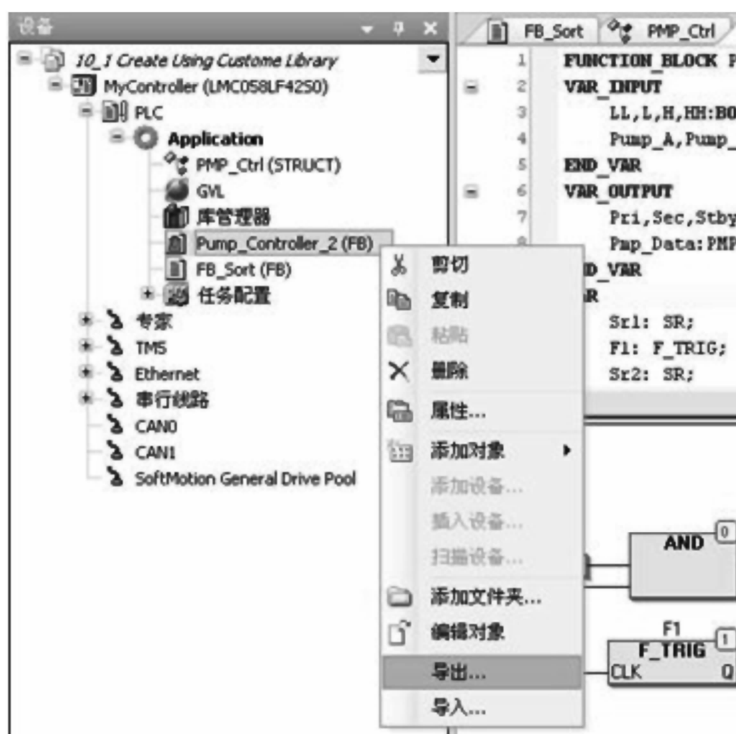


图 10-11　导出功能块

把这个功能导出到计算机上，如图 10-12 所示。

按“确定”后，会弹出导出成功界面，如图 10-13 所示。

同样，我们可以把其他功能块也导出到计算机的指定文件夹里。



图 10-12　导出到计算机上的文件夹



图 10-13　导出成功

我们把这组专用的功能块导出后，就可以建立一个专用的功能库，把这些功能块都放在这个库里。当其他项目需要时，就把这个库安装上，编程时就可以非常方便地使用这些功能了。在建库时，首先使用 SoMachine 的空项目启动一个编辑，如图 10-14 所示。



图 10-14　使用空项目启动

启动后，我们命名一个库文件名，然后选择保存类型为库文件，如图 10-15 所示。



图 10-15　将项目存为库文件

打开库文件程序，点击视图下拉菜单，选择 POU，然后点击“工程”，选择“工程信息”，如图 10-16 所示。

确定后，打开工程信息框，我们可以编辑库文件的所属公司、库文件的标题和版本号以及作者名和库的简要说明，如图 10-17 所示。



图 10-16　选择“工程信息”



图 10-17　填写库文件信息

选中“pump _ control”点击右键打开下拉菜单，选择“导入”，如图 10-18 所示。



图 10-18　导入功能块

找到计算机中导出文件的文件夹，选择要导入的功能块，如图 10-19 所示。

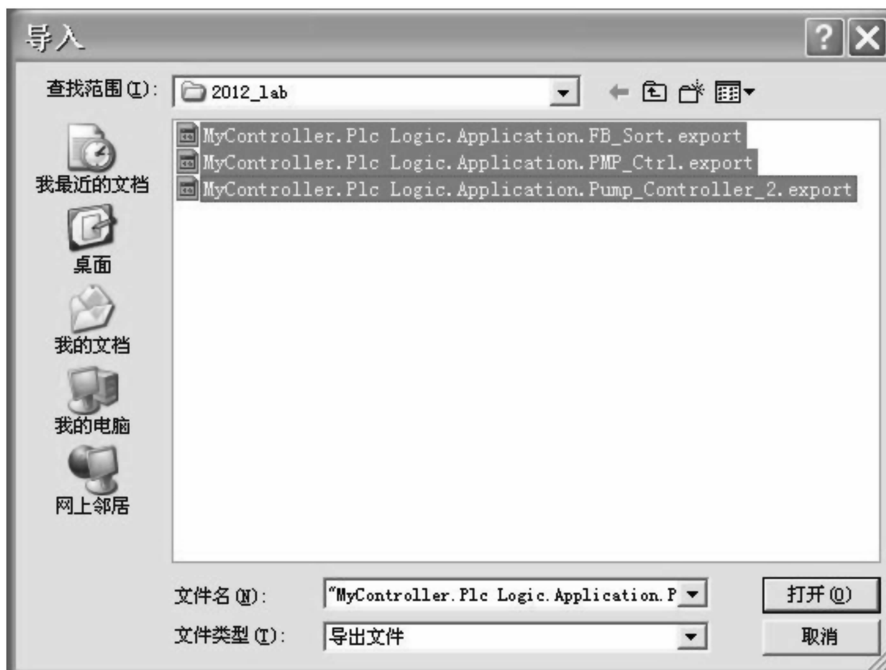


图 10-19 导入编辑好的功能块

导入完成后，选中“pump _ control”点击右键打开下拉菜单，选择添加对象—库管理器，如图 10-20 所示。



图 10-20 添加库管理器

打开库管理器后，选择“添加库”，在添加库页面点击“占位符”，然后选择“占位符名称”和“公司”，如图 10-21 所示。在“Toolbox”下选择 * 意味着最新版本。

然后，再添加标准库到你的库中，如图 10-22 所示。

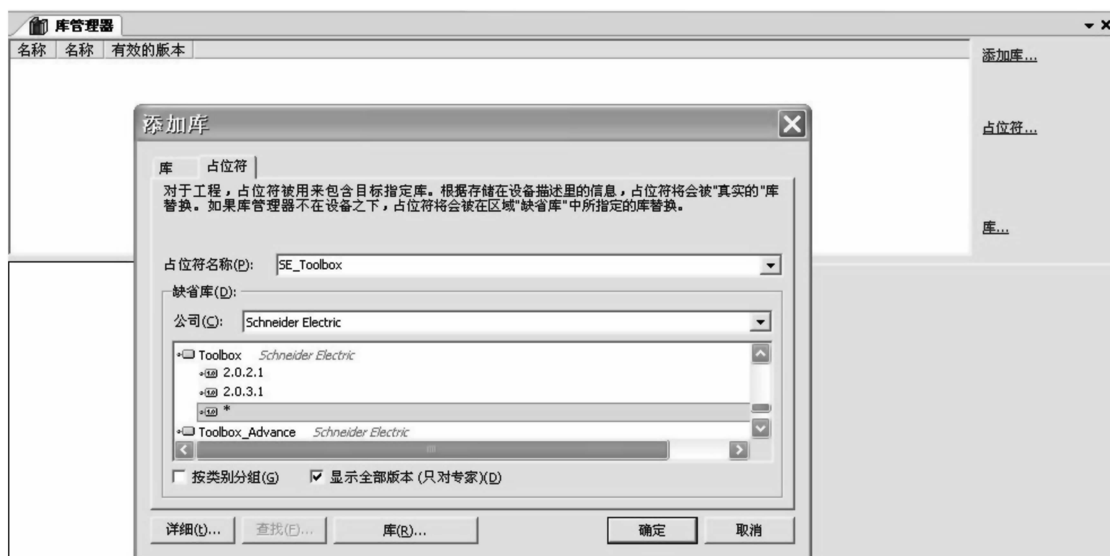


图 10-21　添加库



图 10-22　添加标准库文件

添加完库文件后，我们可以在管理器中看到这些文件，如图 10-23 所示。

点选“编译”，检查所有编辑文件，如图 10-24 所示。

编译成功无错误后，就可以保存这个库文件了，如图 10-25 所示。

这样库文件就建立好了。当编写其他项目要用到这个功能库时，就可以安装并添加。在添加时，首先在这个项目的程序工具栏，选定库，如图 10-26 所示。

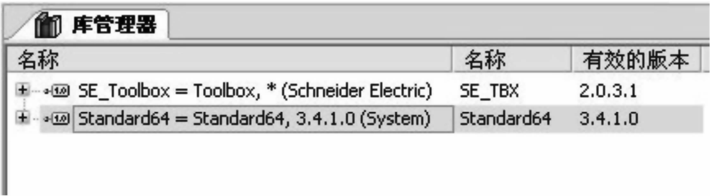


图 10-23　添加的库文件



图 10-24　检查库文件



图 10-25　保存编辑好的库文件

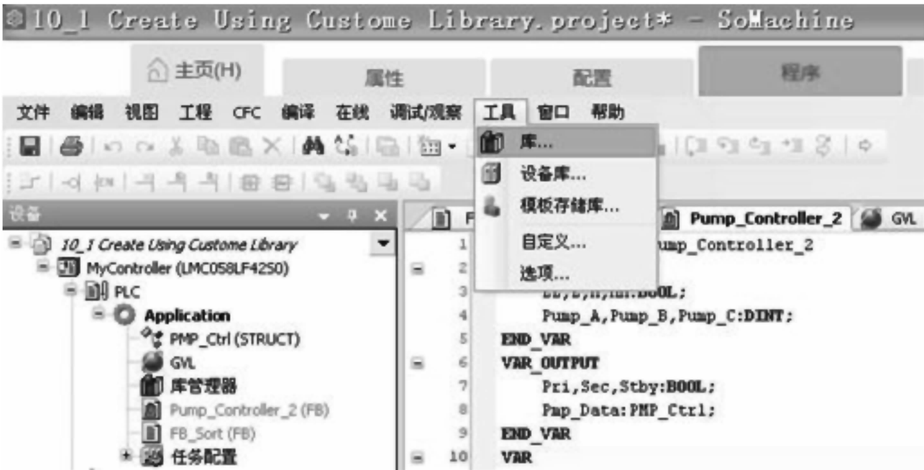


图 10-26　添加用户功能库

确认后，会出现如图 10-27 所示界面。在公司栏目里，通过下拉菜单，选择建库时在工程信息中填写的公司名称，然后对应的库文件名就显示出来了。选择新的版本，然后点击“安装”。

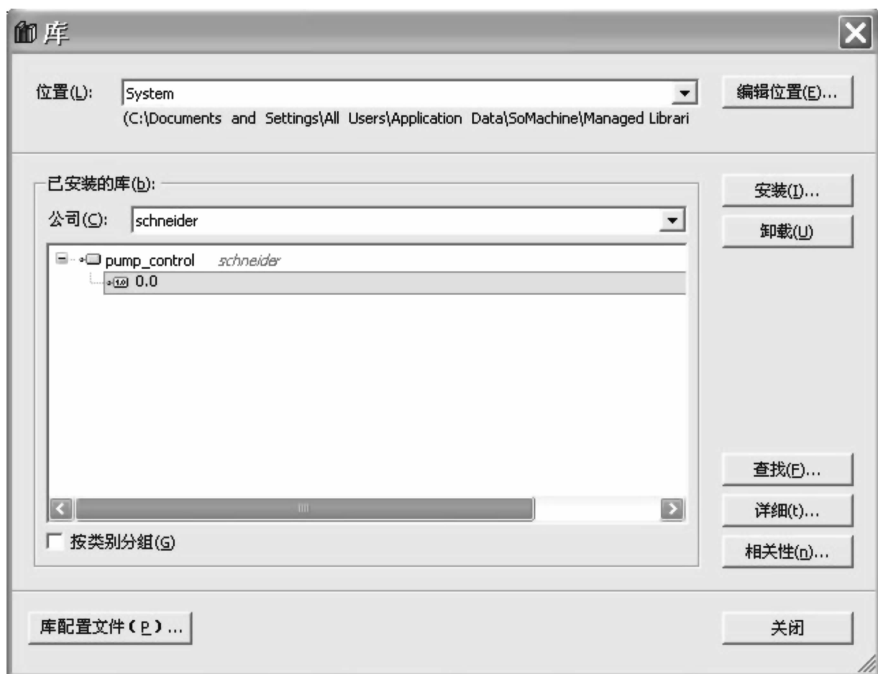


图 10-27　在库中选择要安装的用户库

点击“安装”后，选择库导引到保存库文件的地址，在这个地址下，选择要安装的库，如图 10-28 所示。



图 10-28　选择要安装的库文件

安装好以后，要启用还必须通过库管理器进行添加。如图 10-29 所示，首先点击库管理器，然后点击添加库，这时弹出添加库对话框。在这个框中，选择公司名称，然后选择库的版本号。

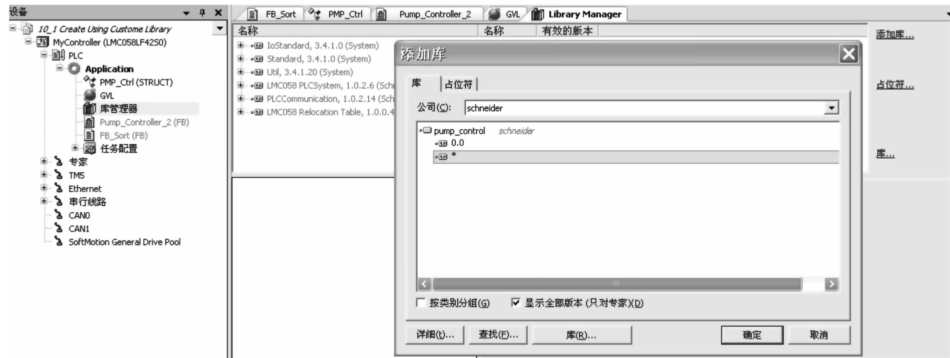


图 10-29　添加用户库

按“确定”后，用户库就被加入到应用。在库管理器中，我们就会看到这个功能库 pump _ control 和它库内的功能块，如图 10-30 所示。

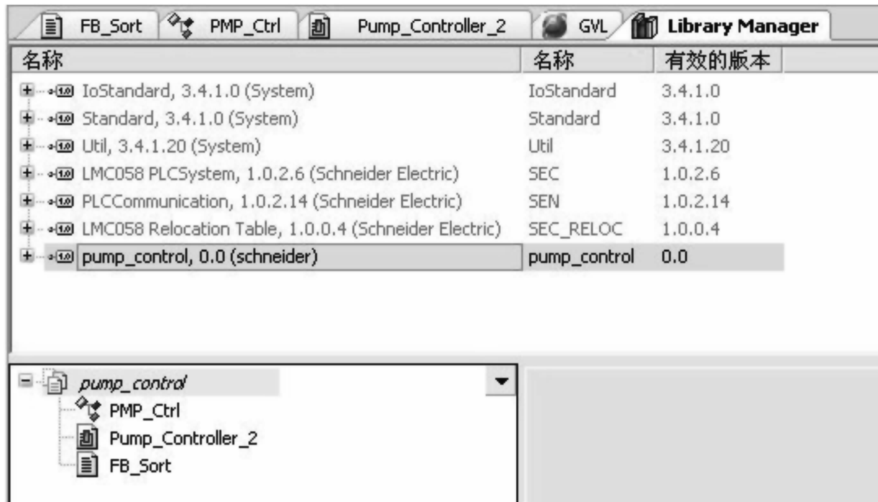


图 10-30　加入的用户库文件

第 11 章　Modbus 通信

11.1　Modbus 通信介绍

在 SoMachine 控制器中，可以通过 Modbus 通信读、写连接在总线上的从设备信息，或作为从设备与主设备通信。这种读、写可以通过两种方式实现：一种是采用 IO 扫描方式，即 IOScanner。在这种方式下，数据是自动交换的，要交换的数据需要事先组态；另一种方式是直接请求，按需进行数据交换，需要在程序组织单元中编辑程序实现。比如借助功能块 read_var 进行设备的数据读出，如图 11-1 所示。

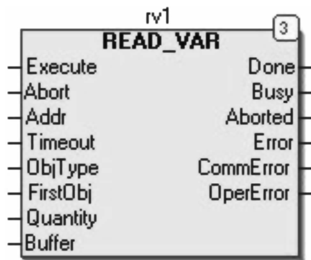


图 11-1　读数据功能块

调出此类功能块，可以在如图 11-2 所示的功能块组找到。

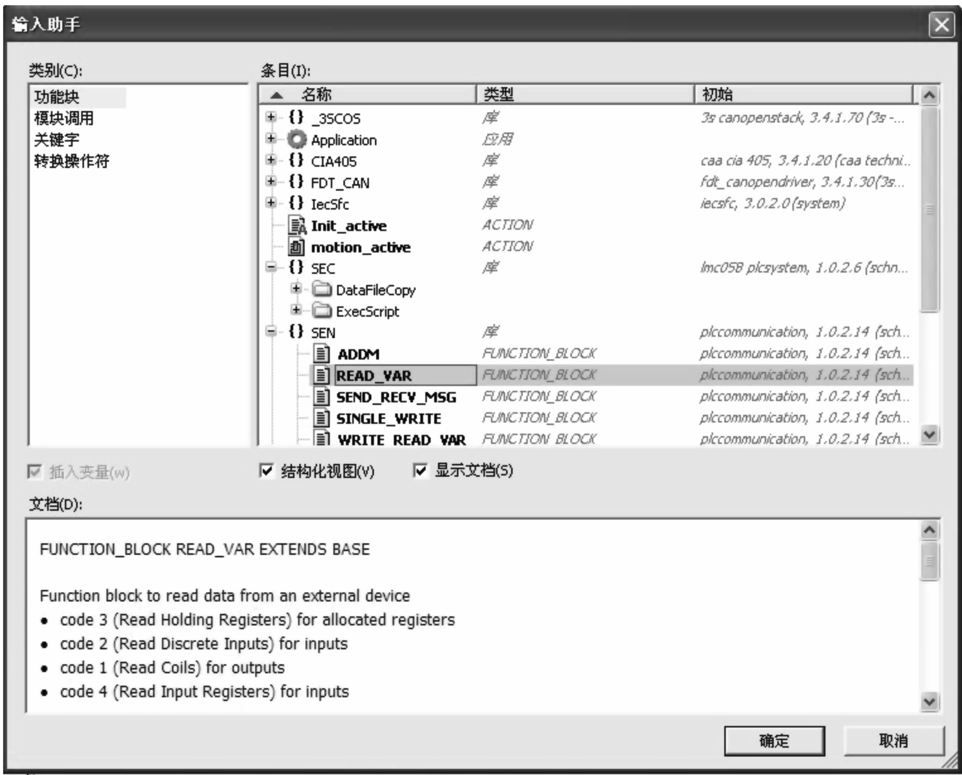


图 11-2　读写数据功能块

11.2　通信的组态

在上一节，我们知道了 Modbus 的两种通信方式，这节将讲述如何构建一个通信。在串行线路中添加通信设备，首先打开控制器硬件平台，如图 11-3 所示，找到串行线路。

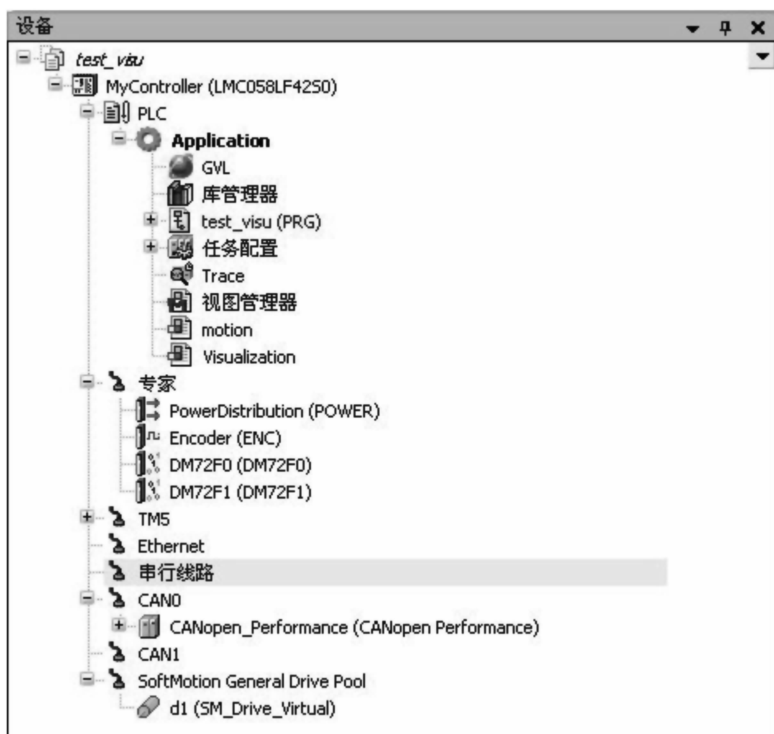


图 11-3　控制器硬件配置

选中串行线路，点击鼠标右键，选择“添加设备”，如图 11-4 所示。

确认添加后，会有添加设备框出现，如图 11-5 所示。

点击现场总线，Modbus 下会有两个选项，如图 11-6 所示，一个是控制器作为主站，连接在总线上的设备都作为从站的总线扫描模式，即 Modbus IOScanner；另一个是控制器可以作为主站，也可以作为从站的数据交换请求模式，即 Modbus_Manager。

根据需要，可选择不同的模式。



图 11-4　硬件组态

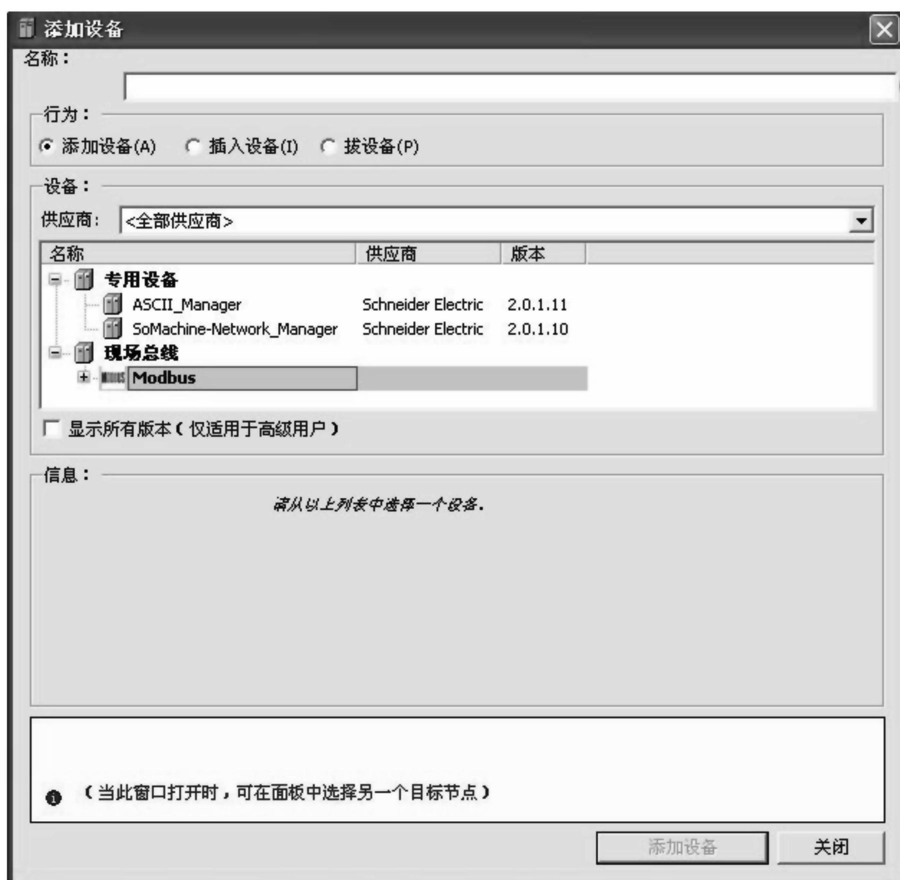


图 11-5　添加设备框



图 11-6　现场总线的硬件组态

11.3　总线扫描模式 IOScanner 组态

当我们选择了 IOScaner 模式时，在串行线路下添加的是 Modbus_IOScanner，如图 11-7 所示。

而这只是添加了一个总线界面。我们还需要把从设备加到这个界面上。如图 11-8 所示，点中 Modbus_IOScanner，按鼠标右键，打开下拉菜单，选择添加设备。

点击添加设备后，弹出添加设备框，如图 11-9 所示。我们可以在名称栏写入要添加的设备名，然后点击“添加设备”。这样在总线接口下就添加了一个从设备 d1，如图 11-10 所示。

如想继续添加从设备，就不必关闭图 11-9 所示的添加设备界面，命名好设备名称，点击“添加设备”即可完成又一次从设备添加。不再添加时，关闭添加设备界面。做好这些硬件配置后，就可以组态从设备的通信了。



图 11-7　选择总线扫描模式



图 11-8　添加从设备



图 11-9　在总线接口下添加设备



图 11-10　加入的从设备 d1

首先，点击串行线路，组态通信口。即在配置栏目下，定义好一个通信参数，如图 11-11 所示。



图 11-11　配置通信口

然后点击“Modbus_IOScanner”，定义传输模式和限定时间，如图 11-12 所示。

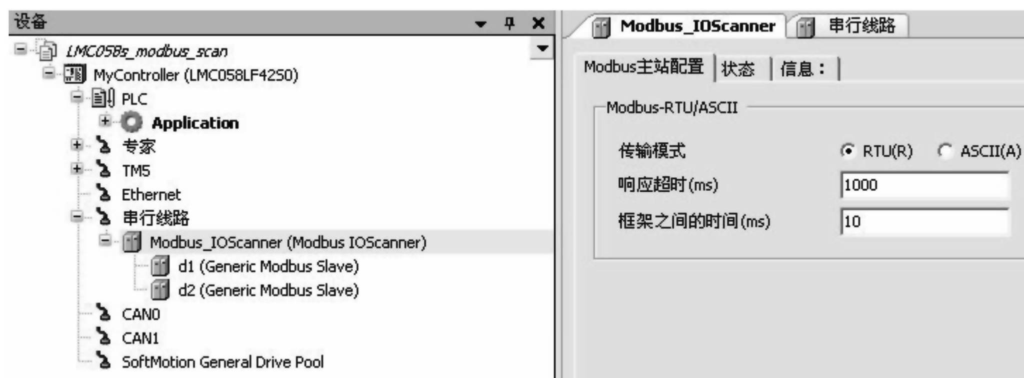


图 11-12　定义传输模式和限定时间

配置完这一步，我们就可以点击从设备 d1，开始对从设备进行组态。首先要定义从设备的地址，如图 11-13 所示。在 Modbus 从站配置栏目下，填写地址。

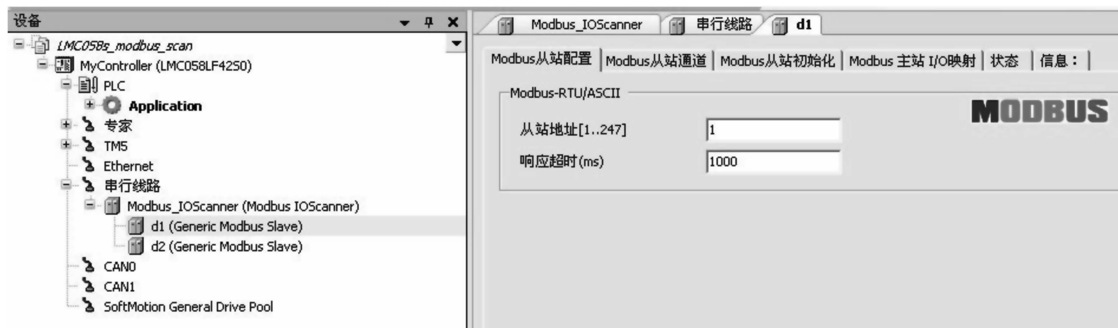


图 11-13　在主站端配置从设备通信地址

同时，我们也要在从设备端配置从设备通信参数，如图 11-14 所示。

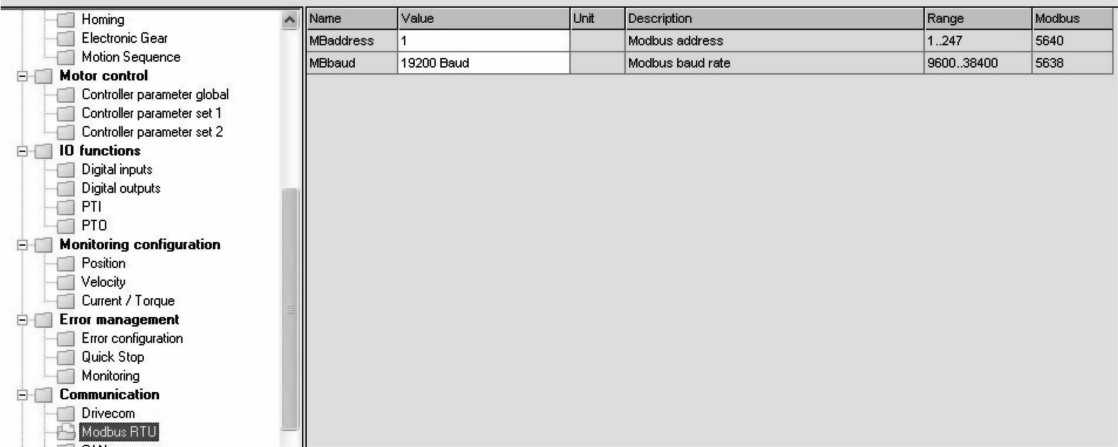


图 11-14　在从站端配置从设备通信参数

然后，点击 Modbus 从站通道，在这个栏目下，选择“添加通道”，如图 11-15 所示。

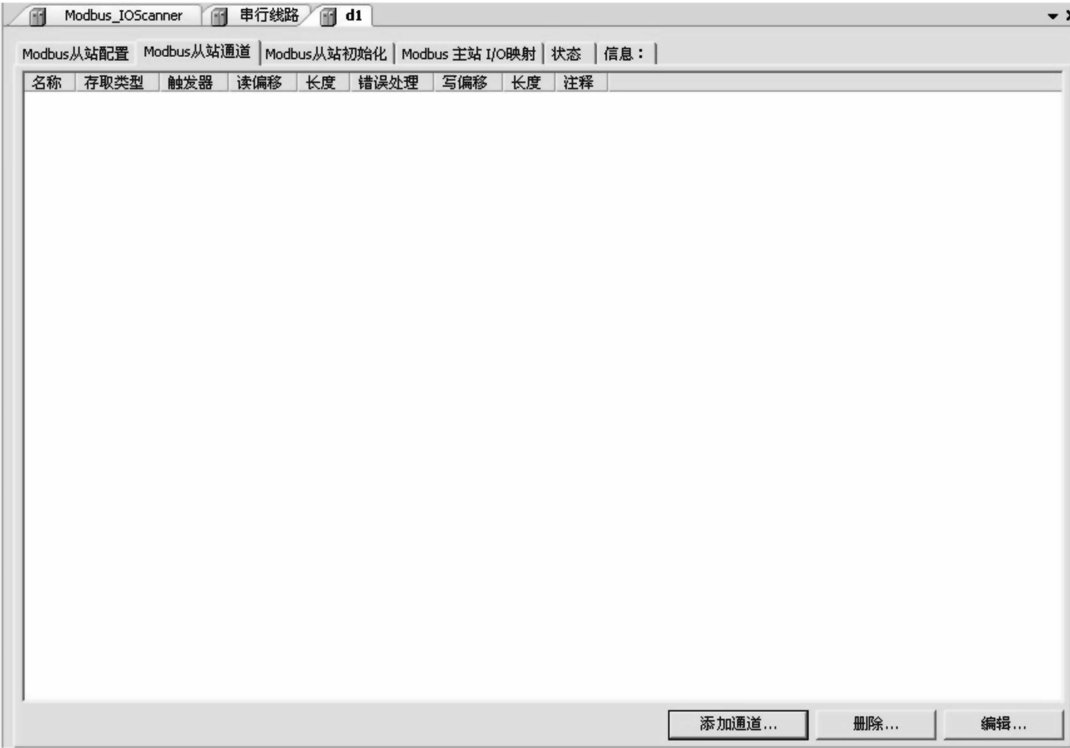


图 11-15　配置从站通道（一）

点击“添加通道”，会弹出通道的配置框，如图 11-16 所示。在这个通道配置框里，可以定义一个通道名。重要的是：通信应用类型（Access Type）有 3 种选择。你可以对从设

备的参数进行读出（函数代码 03）；还可以对从设备的参数进行写入（函数代码 16）；还可以在一个通道对设备同时进行读出和写入操作（函数代码 23）。触发通信的方式（Trigger）可以是周期循环（Cyclic）或上升沿触发（Rising edge）。

当选择读操作时，读寄存器将被激活。当选择写操作时，写寄存器被激活。当选择读、写操作时，读和写寄存器都被激活。

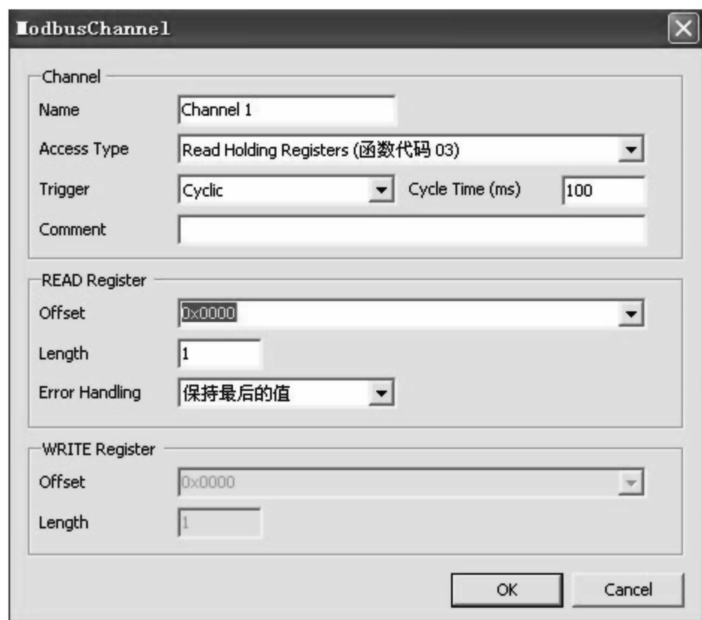


图 11-16　配置从站通道（二）

在读寄存器（READ Register）、写寄存器（WRITE Register）栏目下的 offset 就是总线设备参数的地址，我们可以通过设备手册查到相关参数地址。Length 则是地址所占字长。Offset 的填写可以直接写十进制数，也可以写成 16 进制数，但要加前缀 ox。

例如，我们查到变频器 ATV312 的状态字参数地址为 3201，地址长度为 1。我们就可以选择读操作，把变频器工作状态读出来。配置从站参数读状态如图 11-17 所示。

我们又查到 ATV312 的命令字地址为 8501，命令字后跟一个数据字用来设定变频器的频率，因此字长为 2。8501 的十六进制数为 2135，因此我们也可以按十六进制填写 offset。通过写操作，我们控制变频器按设定频率转动电动机。写操作配置如图 11-18 所示。

配置完以后，我们可以点击 Modbus 主站 I/O 映射，来看读写对应的通道，如图 11-19 所示。

在图 11-19 中，我们看到了映射的对应关系，即变频器的状态字映射给了%IW5，我们通过%IW5 内容变化就可随时知道变频器状态。当我们操作变频器动作时，同样通过对%QW2 赋值命令字，就可完成对变频器的使能，对%QW3 赋值频率，就可让电动机运动起来。

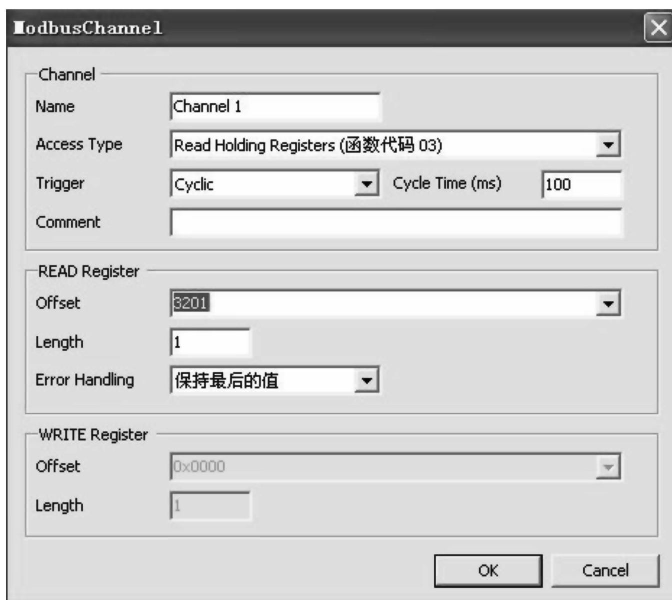


图 11-17　配置从站参数读状态

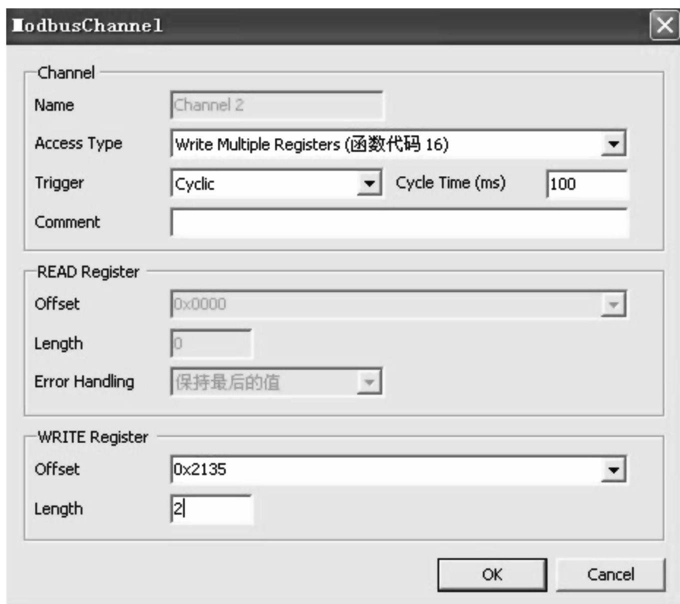


图 11-18　配置从站参数写操作

通过这个例子，我们知道选择总线扫描方式可以不用编写程序，甚至可以不运行 PLC 就可观察设备状态。也就是说：当我们配置好总线扫描方式下的设备参数后，即把控制器的 I/O 资源与从设备联系起来，从设备的一举一动都会在 I/O 资源变化中体现出来，而改变控制器 I/O 资源的状态也会影响到从设备的状态。当然，我们也可以选择在一个通道内同时进行读/写操作，读/写操作配置，如图 11-20 所示。



图 11-19　对应 PLC 输入/输出通道



图 11-20　读/写操作配置

点击 Modbus 主站 I/O 映射，查看与 PLC 资源配置，如图 11-21 所示。



图 11-21　与主站 I/O 映射配置

我们再来看一个实验案例。在这个案例中，将拿伺服驱动器 LEXIUM32C 与运动控制器 LMC058LF42S0 进行 Modbus 通信，通过控制器来观察驱动器的状态变化。我们通过查驱动器手册可知驱动器的控制字 Modbus 参数地址为 6914，如图 11-22 所示。

DCOMcontrol	DriveCom 控制字码 Bit 编码参见操作、运行状态一章 Bit 0：Switch on Bit 1：Enable Voltage Bit 2：Quick Stop Bit 3：Enable Operation Bit 4...6：由运行模式决定。 Bit 7：Fault Reset Bit 8：Halt Bit 9：Change on setpoint Bit 10...15：已保留（必须为 0） 变更的设置将被立即采用。	- - - -	UINT16 读 / 写 - -	Modbus 6914

图 11-22　设备参数地址

查到的状态字地址为 6916，如图 11-23 所示。

_DCOMstatus	DriveCom 状态字 有关 Bit 编码参见机器操作、状态一章 Bit 0...3：状态位 Bit 4：Voltage enabled Bit 5...6：状态位 Bit 7：警告 Bit 8：HALT request active Bit 9：Remote Bit 10：target reached Bit 11：已保留 Bit 12：由运行模式决定 Bit 13：x_err Bit 14：x_end Bit 15：ref_ok	- - - -	UINT16 R/- - -	Modbus 6916

图 11-23　设备状态字地址

根据参数地址，我们在控制器端配置好通道并在线观察映射状态，如图 11-24 所示。

Generic_Modbus_Slave | pou | Serial Line | 任务配置 | MAST | MyController | Modbus_IOScanner

Modbus从站配置 | Modbus从站通道 | Modbus从站初始化 | Modbus 主站 I/O映射 | 状态 | 信息：

通道

变量	映射	通道	地址	类型	缺省值	当前值	准备值	单位	描述
		Channel 1	%IW5	WORD		0			READ 16#1B02 (=06914)
		Channel 1	%IW6	WORD		0			READ 16#1B03 (=06915)
		Channel 1	%IW7	WORD		0			READ 16#1B04 (=06916)
		Channel 1	%IW8	WORD		24625			READ 16#1B05 (=06917)

图 11-24　驱动器初始状态

在这个初始状态，我们看到命令字为“0”，状态字为 24625，转化为十六进制为 16#6031。

当我们在驱动器面板上操作，使电动机使能后，我们看到的状态如图 11-25 所示。

电动机使能后，命令字变为“15”，即 16#0F。状态字变为 16439，即 16#4037。

当在驱动器面板上操作点动 JOG，使电动机向前、向后动作时，映射的状态如图 11-26 所示。

我们看到命令字没变，状态字变为“55”。



图 11-25　驱动器使能后的状态



图 11-26　驱动器 JOG 模式使电动机向前、向后运动时，映射的状态

运行这个案例时，控制器不用工作在运行状态。

11.4　直接请求模式

当在 Modbus 通信中，不需要实时监控从设备状态或控制器本体做从站时，我们可以选择直接请求模式，这种模式按需进行数据交换，从而降低数据通信总线的拥堵，提高通信效率。配置这种模式的通信，需要在串行线路下的添加设备框中，选择添加 Modbus_Manager，如图 11-27 所示。

确认“添加设备”后，我们在设备栏的串行线路下会看到添加的设备名，如图 11-28 所示。

配置完硬件结构后，开始配置通信参数。首先用鼠标双击串行线路，配置通信参数，这与配置扫描模式是一样的。然后，双击设备名 Modbus_Manager，配置通信界面，如图 11-29 所示。通过这个界面，可以定义这个通信界面是主站还是从站。如果定义为主站，则配置到此结束，所有的从设备可以连接在这个端口上，通信的实现，依靠编辑程序并运行来完成。从设备的地址由程序功能块来定义。当选择了从站，则还需要定义这个端口作为从站的地址。

我们配置好 Modbus 通信口后，就可以编写程序来请求与从设备进行通信并进行数据交换。这种通信和数据交换，我们要用到 3 个基本的功能块，它们是：ADDM、READ_VAR、WRITE_VAR，如图 11-30 所示。



图 11-27　添加 Modbus 串行端口下的 Modbus_ Manager

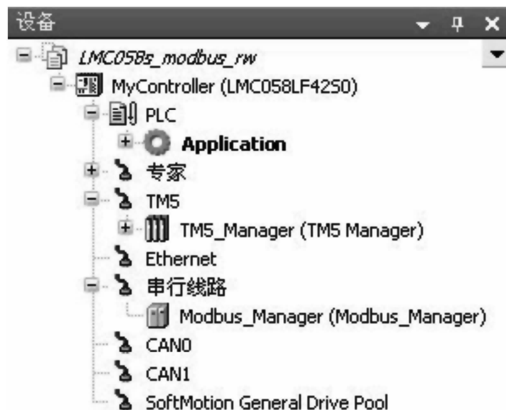


图 11-28　添加的设备 Modbus_Manager



图 11-29　设备通信配置界面

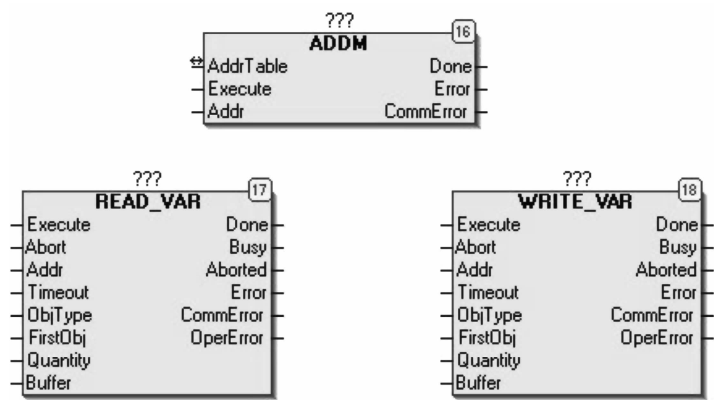
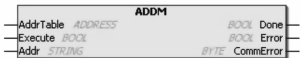


图 11-30　基本的通信功能块

这些功能块可以用于通过 Modbus 的通信，也可以用于通过以太网的通信。本章我们阐述通过 Modbus 的通信，后面章节会阐述通过以太网的通信。在这里先介绍 ADDM 的作用。ADDM 的功能是把显示为字符串的目标地址转化为通信功能块中管脚属性为 ADDRESS 的地址名称内容。读/写功能块通过这个地址名称就可以识别从设备并与之通信。从某种意义上说就相当于把 Addr 的输入字符解码并赋值给 AddrTable 的输入地址名称中。ADDM 的输入/输出定义如图 11-31 所示。

根据这个定义，我们进行功能块应用的输入/输出管脚命名。首先，我们知道 AddrTable 是一个结构变量，属性为 ADDRESS。它只需命名一个变量，而变量的内容是由 Addr 输入解码来赋值的。Execute 是一个开关量，用于启动一次解码。Addr 输入是一个字符串，它基于不同的硬件端口而有所不同。SoMachine 平台的控制器通信端口定义如图 11-32 所示。

图形表示形式



特定于 ADDM 的参数介绍

输入/输出	类型	注释
AddrTable	ADDRESS	这是由功能块填充的 ADDRESS 结构。

输入	类型	注释
Execute	BOOL	在上升沿执行功能
Addr	STRING	要转换为 ADDRESS 类型的 STRING 类地址（参见下面的详细信息）

输出	类型	注释
Done	BOOL	功能成功完成后，Done设置为 TRUE。 注意：当使用 Abort 输入中止操作后，Done 不设置为 1（仅限 Aborted）。
Error	BOOL	当功能由于检测到错误而停止时，Error设置为 TRUE。当检测到错误时，CommError和 OperError包含有关检测到的错误的信息。
CommError	BYTE	CommError包含通讯错误代码。

图 11-31　ADDM 的输入/输出定义

枚举器	值（十六进制）	描述
USBConsole	00	没有可用于通讯交换的 USB 端口
COM1	01	串行口 COM 1（嵌入式串行链路）
COM2	02	串行口 COM 2
EthEmbed	03	嵌入式以太网链路
CANEmbed	04	嵌入式 CANopen 链路
COM3	05	串行口 COM 3

如果安装一个串行 PCI 模块，则无论使用哪个物理 PCI 插槽，串行 PCI 模块链路都是 COM2。

如果安装两个串行 PCI 模块，则插在左侧 PCI 插槽上的串行 PCI 模块是 COM2，插在右侧 PCI 插槽上的串行 PCI 模块是 COM3。

图 11-32　控制器通信端口定义

所以，Modbus 串行地址格式的 Addr STRING 如下：

对于 Modbus 串行寻址，使用通信端口和目标从站地址（0 ~ 247），它们之间用点“.”分隔。

语法格式为：

‘<通信口值>.<从站地址>’

例如，配置的从站在串口 1 下，从站的地址为“1”。

则 Addr 的输入字符为‘1.1’。

ASCII 地址格式的 Addr STRING 如下：

对于 ASCII 寻址，只请求通信端口号。

语法格式为：

‘<通信口值>’

例如，要在串行线路 2 上发送用户定义的消息，使用字符串。

则 Addr 的输入字符为‘2’。

通过 ADDM 功能块定义好通信的从站地址后，就可以通过读/写功能块和这个从站设备进行通信了。

通过在线帮助，我们知道读功能块定义如图 11-33 所示。


READ_VAR：从 Modbus 设备读取数据

READ_VAR：从 Modbus 设备读取数据

功能描述

READ_VAR功能块从采用 Modbus 协议的外部设备中读取数据。

图形表示形式

特定于 READ_VAR 的参数介绍

输入	类型	注释
ObjType	ObjectType	ObjType是要读取的对象的类型（MW、I、IW 和 Q）。
FirstObj	DINT	FirstObj为要读取的第一个对象的索引。
Quantity	UINT	Quantity为要读取的对象的数量： • 1-125: 寄存器（MW 和 IW 类型） • 1-2000: 位（I 和 Q 类型）
Buffer	POINTER TO BYTE	Buffer为将在其中存储对象值的缓冲器的地址。ADR标准功能是定义关联指针的必用功能。（请参见下面的示例。）缓冲器是一个表，用于接收在设备中读取的值。

所有 PLCCommunication 库功能块共用的输入和输出参数会在其他部分介绍。

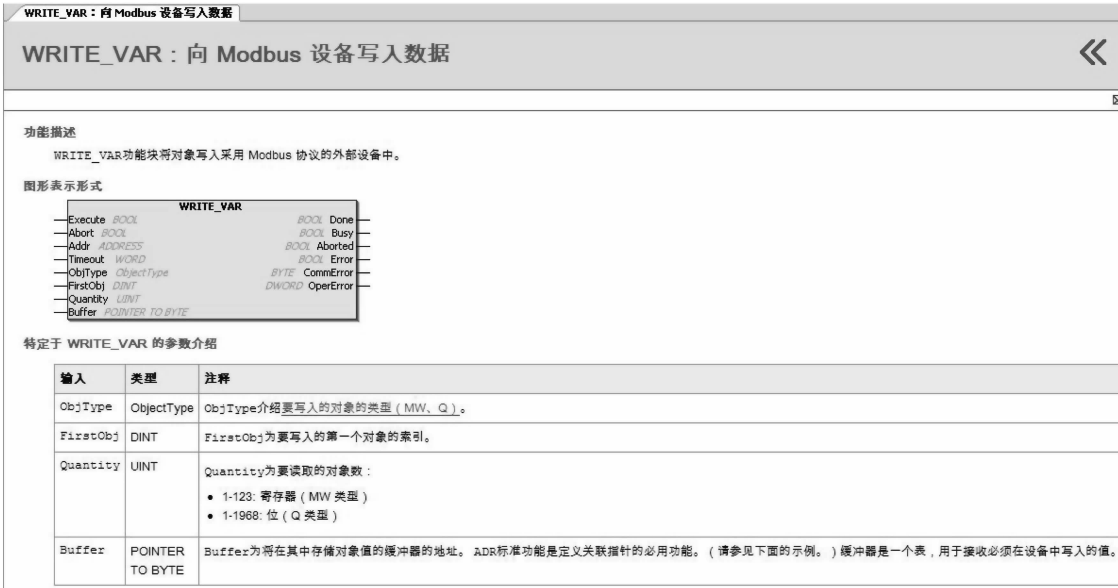
图 11-33　读功能块定义

在这里 Execute 是开关量，每一次开关上升沿启动一次读操作。Abort 退出操作。Addr 是从设备地址名，来自 ADDM 的定义。Timeout 是读操作限定时间，它的单位是 100ms。ObjType 是一个枚举变量，它用来定义读取对象的类型。例如，我们定义读取对象为 ObjectType. MW。FirstObj 是要读取从设备参数的 Modbus 首地址。Quantity 是读取对象的数量。Buffer 是指定的一个缓冲器的位置，用来存放读取的设备参数值。

同理，我们可知道写功能块的定义如图 11-34 所示。

在这里 Execute 是开关量，每一次开关上升沿启动一次写操作。Abort 退出操作。Addr 是从设备地址名，来自 ADDM 的定义。Timeout 是写操作限定时间，它的单位是 100ms。ObjType 是一个枚举变量，它用来定义写入对象的类型。例如，我们定义写入对象为 ObjectType. MW。FirstObj 是要写入从设备参数的 Modbus 首地址。Quantity 是写入对象的数量。Buffer 是指定的一个缓冲器的位置，用来存放写入的设备参数值。

根据这些读写定义和操作规则，我们就可以编辑一段程序来实现对从设备的读/写操作。首先，我们调入功能块 ADDM，定义从设备地址。控制器的通信口为“1”，从站设备地址也是“1”。因此，程序如图 11-35 所示。


WRITE_VAR：向 Modbus 设备写入数据

WRITE_VAR：向 Modbus 设备写入数据

功能描述

WRITE_VAR功能块将对象写入采用 Modbus 协议的外部设备中。

图形表示形式

特定于 WRITE_VAR 的参数介绍

输入	类型	注释
ObjType	ObjectType	ObjType介绍要写入的对象的类型（MW、Q）。
FirstObj	DINT	FirstObj为要写入的第一个对象的索引。
Quantity	UINT	Quantity为要读取的对象数： • 1-123: 寄存器（MW 类型） • 1-1968: 位（Q 类型）
Buffer	POINTER TO BYTE	Buffer为将在其中存储对象值的缓冲器的地址。ADR标准功能是定义关联指针的必用功能。（请参见下面的示例。）缓冲器是一个表，用于接收必须在设备中写入的值。

图 11-34　写功能块定义

```
PROGRAM POU
VAR
    d1_add: ADDM;
    d1_table: ADDRESS;
    k1: BOOL;
    addr_done: BOOL;
    d1 read: READ VAR;
```



图 11-35　地址定义

定义好地址后，我们调入读/写功能块，并定义好输入和输出，如图 11-36 所示。

编辑完这些程序，就可以运行控制器执行这些程序。首先按下 k1，使地址解码赋值，没有错误，则 addr_done 输出高电平。接着可查出要读出的参数地址，比如要看变频器 ATV312 的状态，它的参数地址是 3201，那么就把 3201 赋值给 d1_readcmd 即可。赋值完毕，按下 k2，读出的状态就被放在缓冲器里，并且输出给 r1、r2。

当要对从设备操作时，我们就可对 d1_wrcmd 赋命令地址值，命令后面的数据值放在 SendBuffer 缓冲器里。例如，变频器的使能启动命令地址是 8501，转动频率值地址是 8502。因此，把 8501 赋值给 d1_wrcmd，然后把命令赋值给 wcmd_para，把频率值赋值给 wcmd_para2，这样缓冲器里就存好了要写入到从设备的参数。按下 k3，完成写入。

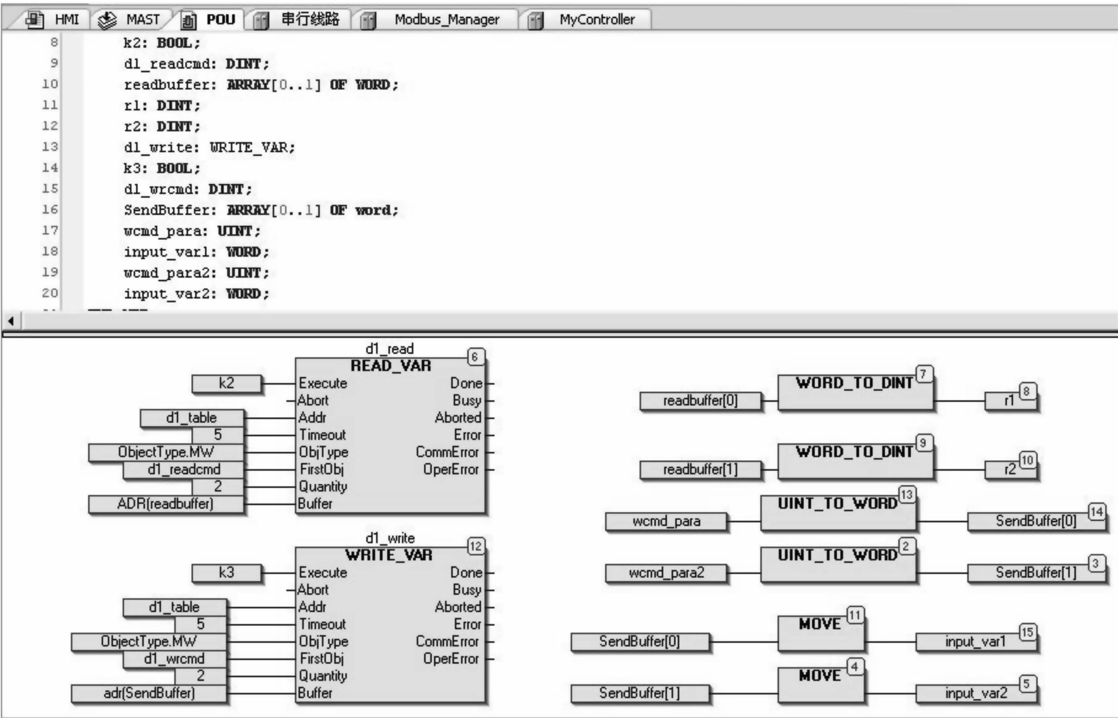


图 11-36　读/写功能块的编写

利用这个程序，我们做如下案例。

复位 LEXIUM32C 驱动器错误

采用 Modbus 控制驱动器输出点开关 DQ0，来复位驱动器。

查 I/O 置位命令地址如图 11-37 所示。

Parameter name HMI menu HMI name	Description	Unit Minimum value Factory setting Maximum value	Data type R/W Persistent Expert	Parameter address via fieldbus
IO_DQ_set	Setting the digital outputs directly Write access to output bits is only active if the signal pin is available as an output and if the function of the output was set to 'Available as required'. Coding of the individual signals: Bit 0: DQ0 Bit 1: DQ1 Bit 2: DQ2 Bit 3: DQ3 Bit 4: DQ4	- - - -	UINT16 R/W - -	Modbus 2082

图 11-37　I/O 置位命令地址

01：DQ0 输出

操作界面如图 11-38 所示。

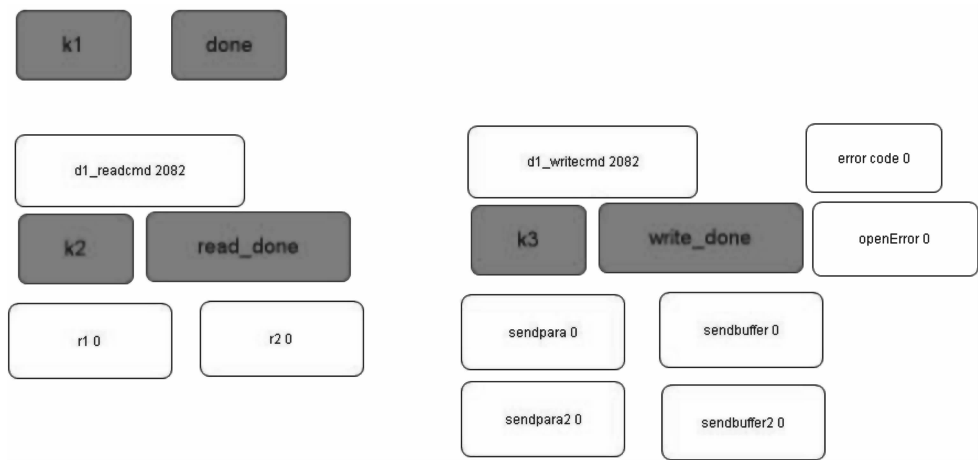


图 11-38　与程序对应的操作界面

首先读出输出点状态，如图 11-39 所示。

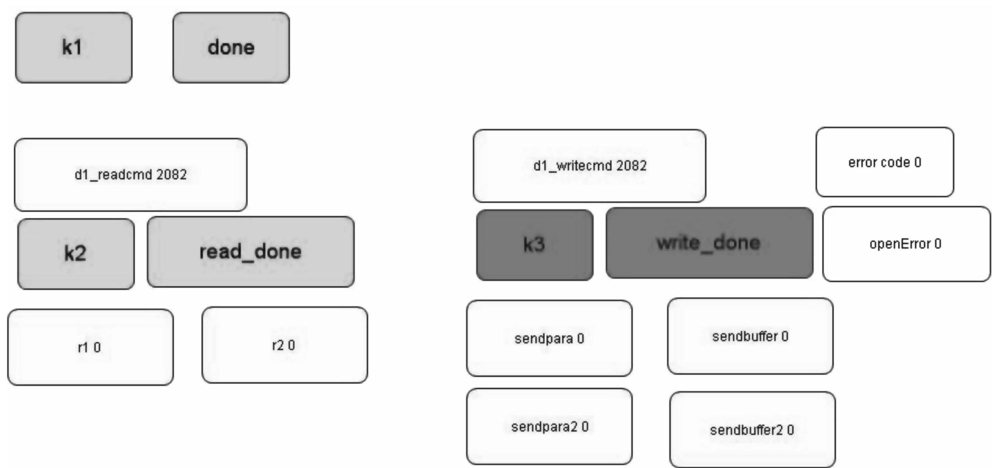


图 11-39　读出操作界面状态

读出值为 00，说明没有输出。

使 DQ0 输出，采用写输出。写入操作界面状态如图 11-40 所示。

注意：sendpara = 0；sendpara2 = 1。

我们再来读一下 DQ0 状态，如图 11-41 所示。

可以看到输出为 01。

这时，DQ0 输出。

在驱动器中，把 DI3 设为复位，DQ0 的输出接入 DI3 输入，即可完成通过总线使驱动器复位。设置复位点如图 11-42 所示。

程序见附件 LMC058s_modbus_rw。

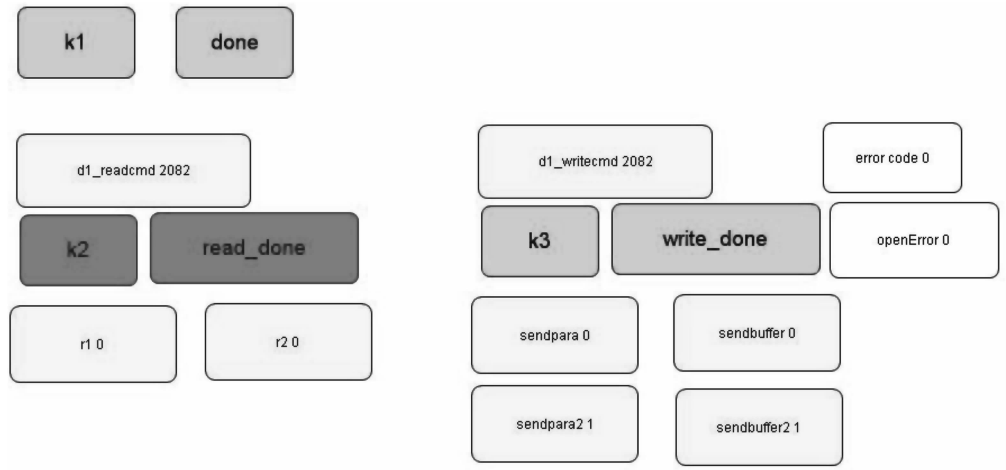


图 11-40　写入操作界面状态

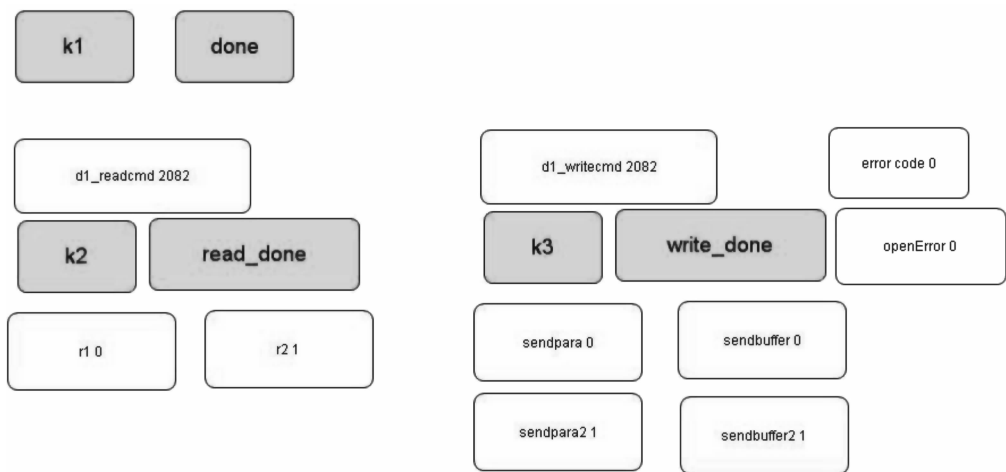


图 11-41　DQ0 状态

Position Register
Operation configuration
Jog
Homing
Motor control
Controller parameter global
Controller parameter set 1
Controller parameter set 2
IO functions
Digital inputs
Digital outputs

Name	Value	Unit	D
IOfunct_DI0	Freely Available		F
IOfunct_DI1	Freely Available		F
IOfunct_DI2	Freely Available		F
IOfunct_DI3	Fault Reset		F
DI_0_Debounce	1.50 ms		D
DI_1_Debounce	1.50 ms		D
DI_2_Debounce	1.50 ms		D
DI_3_Debounce	1.50 ms		D

图 11-42　设置复位点

第 12 章　CANopen 通信

在 SoMachine 控制平台设计一个复杂的机器控制系统，最优的结构就是使用 CANopen 总线，把诸多的控制元素诸如变频器、伺服驱动器、传感器、开关按钮等通过一条电缆连接起来，共享各个控制单元的状态信息、控制指令等，如图 12-1 所示。大量使用 CANopen 总线不仅由于其通信速度大大高于 Modbus，更是由于其良好的抗干扰能力和简单的物理安装结构及组态方式。本章主要讨论 CANopen 总线的特性及应用。

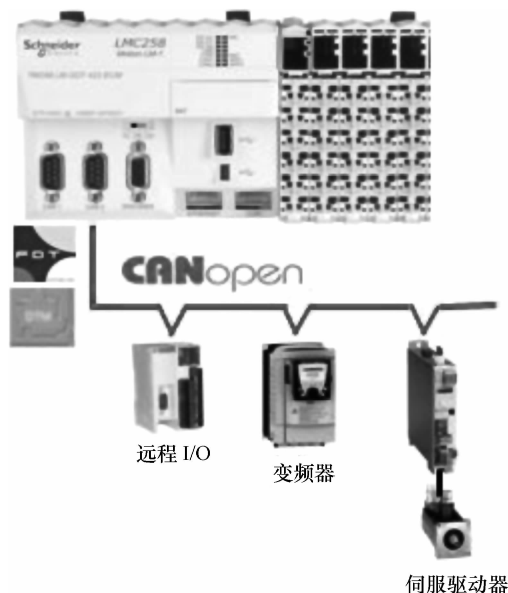


图 12-1　带有 CANopen 总线的控制系统

12.1　CANopen 总线基础

CANopen 是一种架构在控制局域网路（Control Area Network，CAN）上的高层通信协定，包括通信子协定及设备子协定常在嵌入式系统中使用，也是工业控制常用到的一种现场总线。

CANopen 实际作了 OSI 模型中的网络层以上（包括网络层）的协定。CANopen 标准包括寻址方案、数个小的通信子协定及由设备子协定所定义的应用层。CANopen 支持网络管理、设备监控及节点间的通信，其中包括一个简易的传输层，可处理资料的分段传送及其组合。一般而言，资料链接层及实体层会用 CAN 来实际操作。

基本的 CANopen 设备及通信子协定定义在 CAN in Automation（CiA）标准 301 中。针对个别设备的子协定以 CiA 301 为基础再进行扩充。如针对 I/O 模组的 CiA401 及针对运动控

制的 CiA402。其总线架构如图 12-2 所示。

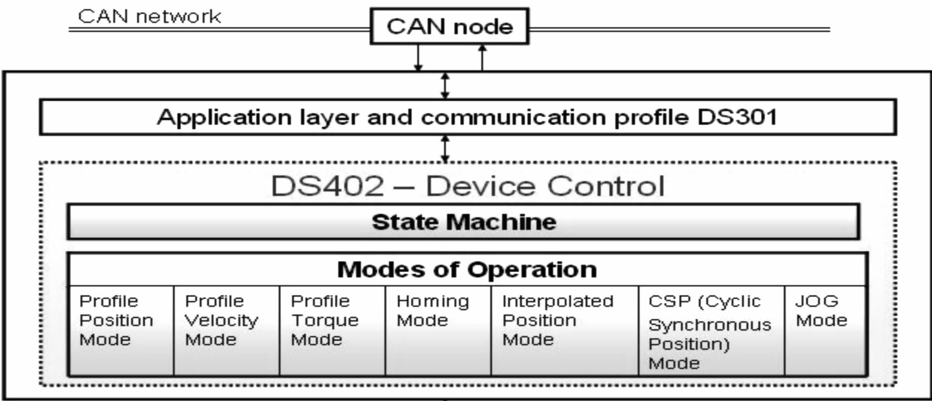


图 12-2 CANopen 总线架构

CANopen 凭借其可靠、实时、灵活、经济的特点而被广泛应用，并且已成为国际标准。CANopen 协议是在现场总线 CAN-BUS 之上定义的一套应用层协议，其核心是对象字典及多种通信方式。对象字典是应用单元和通信单元之间的接口，实际上是设备的所有参数列表。应用单元和通信单元都可访问这个参数列表。对象字典中的条目（对象或参数）通过一个 16 位索引（Index）和一个 8 位子索引（Subindex）进行识别或定位，用户可对条目进行读或写。对象字典的分配见表 12-1。

表 12-1 对象字典的分配

索　引	对　象	解　释
0x0000	保留	
0x0001-0x009F	数据类型区	定义各种用到的数据类型，如字、字节、整数等
0x00A0-0x0FFF	保留	
0x1000-0x1FFF	通信对象	对要进行通信的对象做描述
0x2000-0x5FFF	制造商特定的对象	制造商特定对象的描述
0x6000-0x9FFF	标准化设备对象	对标准化对象的描述
0xA000-0xFFFF	保留	

对象字典描述了对象设备的所有功能。例如伺服驱动器的索引 6060：0 定位到了设备的运行模式，后面跟的数据确定了运行模式，如图 12-3 所示。

每个设备制造商会为自己的设备做好一个 ASCII 码格式的对象字典，即电子数据表（Electronic Data Sheet，EDS）文件。这个电子数据表可以导入到控制器，实现控制器与设备的 CANopen 通信。好了，让我们再来梳理一下有关的 CANopen 通信要用到的概念和名称。

1. 设备通信

通信单元处理和网络上其他模组通信所需要的通信协定。设备的起动及重置由状态机（State Machine）控制。状态机需包括以下几个状态：初始化（Initialization）、预运行（Pre-operational）、运行（Operational）及停止（Stopped）。当接收到网络管理（NMT）通信对象，

DCOMopmode	运行模式 **-6 / Manual Tuning / Autotuning**：手动调整/自动调整 **-1 / Jog**：Jog **0 / Reserved**：已保留 **1 / Profile Position**：Profile Position **3 / Profile Velocity**：Profile Velocity **4 / Profile Torque**：Profile Torque **6 / Homing**：Homing 变更的设置将被立即采用。	- -6 - 6	INT8 INT16 读 / 写 - -	CANopen 6060:0h Modbus 6918

图 12-3　对象字典定位设备运行模式

状态机会转换到对应的状态。对象字典（Object Dictionary）是一个有 16 位元索引（Index）的变量阵列。每个变量可以（但非必须）有 8 位元的子索引（Subindex）。变量可用来调整设备的组态，也可以对应设备量值的数据或设备的输出。当状态机设定为 Operational 之后，设备的应用（Application）部分就会实现设备预期的功能。此部分可以由对象字典中的变量调整其设定，而数据由通信层传输或接收。

2. 对象字典

CANopen 设备都需要具备对象字典，用来设定设备组态及进行非即时的通信。对象字典的入口定义如下。

索引（Index）：对象 16 位元的位址。对象名称（Object name）：一个代表对象的符号类型，可以是阵列、纪录或只是一个变量。名称（Name）：描述此入口的字串。形态（Type）：变量的数据形态。属性（Attribute）：提供此入口是否可读/可写的数据，有下列四种：可读/写、只读、只写、只读常数。必须（Mandatory）/可选（Optional）字段定义属于特定设备规范下的设备，是否必须实现某些对象。在 CANopen 标准中定义了对象字典中的基本资料形态，包括逻辑值、整数及浮点数。也定义了复合对象，如阵列、记录及字串。复合对象用一个 8 位元的数值作为其子索引（Subindex）。记录或阵列中子索引 0 的位置记录此数据结构的元素个数，资料型态为 UNSIGNED8。

例如在 CiA301 标准中，设备通信的参数放在索引范围 0x1000 ~ 0x1FFF（通信行规区）。此区域的前几项见表 12-2。

表 12-2　通信定义区

索引	对象名称	名称	形态	属性	M/O
0x1000	变量	设备类型	UNSIGNED32	只读	M
0x1001	变量	错误寄存器	UNSIGNED8	只读	M
…					
0x1008	变量	工厂设备名	字符串	常数	O
…					

若配合适当的工具，可以用编辑 EDS 档案的方式规划一个设备，并且将变量的数值上传到设备中。EDS 档案的格式通常会是 INI 文件。

3. 通信对象

CANopen 的物理层 CANbus 每次传送的资料量不大，其中包括 11 位元的 ID、远端传输

请求（RTR）位元及大小不超过 8 位元的数据。CANopen 将 CANbus 11 位元的 ID 分为 4 位元的功能码及 7 位元的 CANopen 节点 ID。7 位元的 ID 共有 128 种不同的组合，其中 ID0 不使用，因此一个 CANopen 网络上最多允许 127 台设备。CANbus 在 CAN 2. 0 B 规格中允许 29 位元的 ID，因此若配合 CAN 2. 0 B 使用，CANopen 网络上可以超过 127 台设备，不过在实际运用中，大多数的 CANopen 网络上设备数量均低于此数值。

CANopen 将 CANbus 的 11 位元 ID 称为通信对象 ID（COB-ID）。当传输数据出现碰撞时，CANbus 的仲裁机制会使 COB-ID 最小的信息继续传送，不用等待或重传。COB-ID 的前 4 个位元是 CANopen 的功能码，因此数值小的功能码表示对应的功能重要，允许的延迟时间较短。

表 12-3 是一个标准的 CANopen 通信帧。

表 12-3　CANopen 通信帧

	功能码	节点 ID	RTR	数据长度	数据
长度	4 位元	7 位元	1 位元	4 位元	0-8 字节

在 CANopen 标准中，部分 COB-ID 被保留作网络管理及 SDO 通信用。而在设备初始化后，有些功能码和 COB-ID 会对应到标准的功能，不过后续仍可以规划为其他用途。

4. 通信模型

CANopen 设备间的通信可分为以下 3 种通信模型。

在 master/slave 模型中，一个 CANopen 设备为主站（master），负责传送或接收其他设备从站（slave）的数据。NMT 协定就使用了 master/slave 模型。客服（client/server）模型定义在 SDO 协定中，SDO 客户将对象字典的索引及子索引传送给 SDO 服务员，因此会产生一个或数个需求数据（对象字典中，索引及子索引对应的内容）的 SDO 封包。生产者/消费者（producer/consumer）模型用在 Heartbeat and Node Guarding 协定。由一个生产者送出数据给消费者，同一个生产者的数据可能给一个以上的消费者。又可分为两种：推送方式（push-model）：生产者会自动送出数据给消费者；拉进方式（pull-model）：消费者需送出请求信息，生产者才会送出数据。

5. NMT 协定

NMT（网络管理，Network Management）协定会定义（设备内部）状态机的状态变更命令（如起动设备或停止设备）、侦测远端设备 bootup 及故障情形。

NMT master 使用的模组控制协定可变更设备的状态。其 COB-ID 为 0，其功能码及节点 ID 均为 0，因此网络上的所有节点均会处理这个信息。在此信息的数据部分会有此信息实际针对节点的 ID，此 ID 也可为 0，表示所有节点都要变更为指定的状态。

使用 CIA405. NMT 功能块可以从控制器应用程序控制 CANopen 设备的 NMT 状态。例如，我们用此功能块复位 CAN 总线或总线上的某个节点。

功能块通过执行对 CANopen 目标设备的 NMT 服务请求，来执行请求的 NMT 状态转换。

CIA405. TRANSITION_ STATE ENUM

NMT 状态机描述主要操作中的 NMT 从站的初始化和状态。

图 12-4 显示的是 NMT 状态、关联的可用通信对象（PDO、SDO、SYNC、EMCY 和 NMT）和 5 种状态转换（A 到 E）。

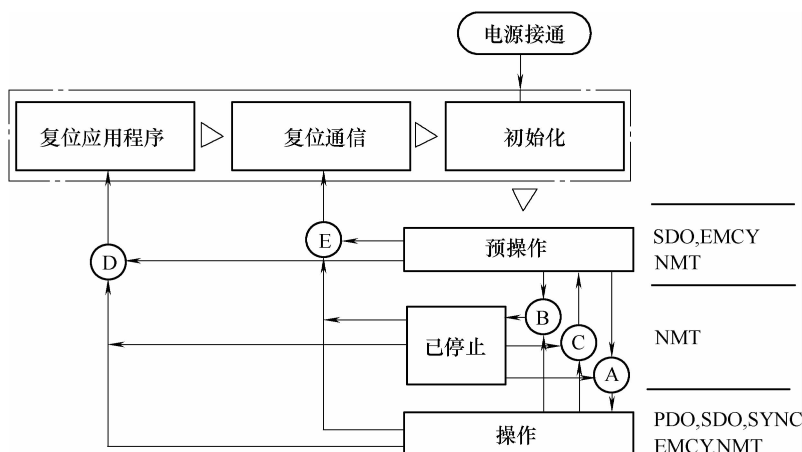


图 12-4　网络管理状态及其转换

CIA405. TRANSITION_ STATE 枚举类型包含表 12-4 中介绍的 NMT 状态转换命令。

表 12-4　CIA405. TRANSITION_ STATE 的枚举变量及功能

枚举器	值(十六进制)	说　明
STOP_REMOTE_MODE	0004	切换到停止状态 (转换 B)
START_REMOTE_MODE	0005	切换到正常操作状态 (转换 A)
REBET_MODE	0006	切换到复位应用程序状态。加载设备配置文件的已保存数据,并自动从复位通讯切换到预操作状态 (特换 D)
REBET_COMMUNICATION	0007	切换到复位通讯状态。加载通讯配置文件的已存储数据,并自动切换到预操作状态 (转换 E)
ENTER_PRE_OPERATIONAL	007F	切换到预操作状态 (转换 C)
ALL_EXCEPT_NMT_AND_BENDER	0800	未实现(无效参数)

6. 心跳协定

心跳协定（Heartbeat protocol）是用来监控网络中的节点及确认其正常工作。心跳信息的生产者（一般是 slave 设备）周期性地送出功能码 1110，ID 为本身节点 ID 的信息，信息的资料部分有一个表示节点状态的位元。而心跳信息的消费者负责接收上述资料，若在指定时间（于设备的对象字典中定义）内，消费者均未收到信息，可采取相关行动（例如显示错误或重置该设备）。

其格式为：COBID + DATA（status of node）CANopen 设备需要在 bootup 时自动从 Initializing 状态切换至 Pre-operational 状态，设备会在切换完成后送出一个心跳信息，这就是心跳

协定。

有一种 pull model 的 NMT 协定，称作节点监控（Node guarding）协定，也可以作从机的监控。

7. 服务数据对象协定

服务数据对象（SDO）可用来存取远端节点的对象字典，读取或设定其中的数据。提供对象字典的节点称为 SDO server，存取对象字典的节点称为 SDO client。SDO 通信一定由 SDO client 开始，并提供初始化相关的参数。

在 CANopen 的术语中，上传是指由 SDO server 中读取数据，而下载是指设定 SDO server 的数据。

8. 过程数据对象协定

过程数据对象（PDO）协定可用来在许多节点之间交换即时的数据。可透过一个 PDO ，传送最多 8 字节（64 位元）数据给一设备，或由一设备接收最多 8 字节（64 位元）的数据。一个 PDO 可以由对象字典中几个不同索引的资料组成，规划方式则是透过对象字典中对应 PDO 配置及 PDO 参数的索引。

PDO 分为两种：传送用的 TPDO 及接收用的 RPDO。一个节点的 TPDO 是将资料由此节点传输到其他节点，而 RPDO 则是接收由其他节点传输的资料。一个节点分别有 4 个 TPDO 及 4 个 RPDO。

PDO 可以用同步或异步的方式传送：同步的 PDO 是由同步 SYNC 信息触发，而异步的 PDO 是由节点内部的条件或其他外部条件触发。例如若一个节点规划为允许接受其他节点产生的 TPDO 请求，则可以由其他节点送出一个没有数据但有设定 RTR 位元的 TPDO（TP-DO 请求），使该节点送出需求的数据。

借由 RPDO 也可以使两种设备同时起动。

除了定义对象字典，CANopen 协议还定义了网络物理结构，如图 12-5 所示。即连接各个设备站点的电缆是带屏蔽的双绞线，传输线的终端电阻为 120Ω。在整个网络中包括主站在内的站点数，最多不能超过 127 个。电缆的信号为 CAN_H、CAN_L 和 CAN_GND。在同一网络内，各个站点的通信速率要求配置一样。通信速率根据传输距离的大小而有不同。请参见表 12-5 电缆长度和速率的配置。

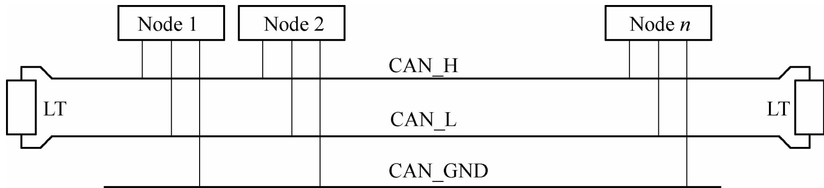


图 12-5　CANopen 总线的物理结构

表 12-5　电缆长度和速率的配置

波特率		1Mbit/s	800kbit/s	500kbit/s	250kbit/s	125kbit/s	50kbit/s	20kbit/s	10kbit/s
最大电缆长度	m	4	25	100	250	500	1000	2500	6000
	ft(英尺)	13.12	82.02	328.08	820.20	1640.41	3280.83	8202.07	16404.15

在控制器上的 CANopen 总线口被设计为 D 型 9 针口，如图 12-6 所示。在运动控制器中有两个 CANopen 总线口，一个是通用总线口，一个是可以做多轴同步的总线口，我们称为 CANmotion 总线口。

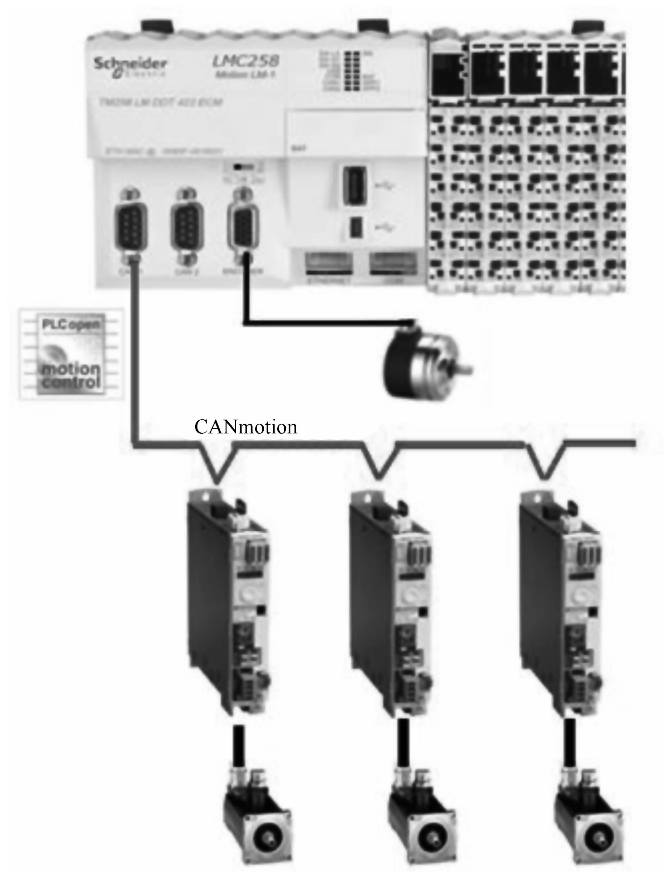


图 12-6　CANopen 在控制器上的端口

CANopen 接线的规定如图 12-7 所示。D 型 9 针插头的 CANopen 接线定义见表 12-6。

通常 CANopen 现场总线，在现场中使用带有 D 型插头的电缆，它的优点是抗干扰性强，可靠性高。在控制柜中采用 RJ45 电缆进行连接，它的的优点是布线简单又快捷。驱动器上的 CANopen 总线 RJ45 端口定义如图 12-8 所示。

注意：使用带有 RJ45 连接器的电缆时，最大总线长度减半。

当波特率为 1Mbit/s 时，传输线就限制为 3m。

RJ45 接头定义见表 12-7。

这些定义都是 CANopen 总线的标准，这使得各个设备制造商提供的设备无论从软件配置还是硬件接线都有了统一的标准，总线互联变得非常方便。

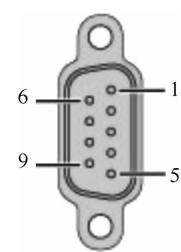


图 12-7　CANopen 接线的规定

表 12-6　D 型 9 针插头的 CANopen 接线定义

引脚编号	信　号	描　述	引脚编号	信　号	描　述
1	N. C.	保留	6	GND	0Vdc
2	CAN_L	CAN_L 总线(低)	7	CAN_H	CAN_H 总线(高)
3	CAN_GND	CAN 0Vdc	8	N. C.	保留
4	N. C.	保留	9	N. C.	保留
5	CAN_SHLD	N. C.			

N. C.：未连接。

该屏蔽已连接到引脚 6，即 0Vdc 引脚。

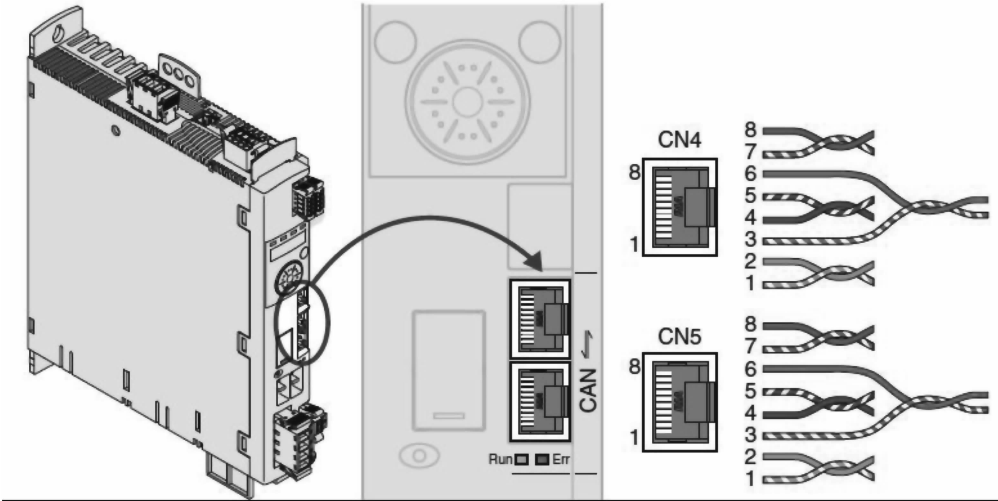


图 12-8　驱动器 CN4 和 CN5 上的 CANopen 总线 RJ45 端口定义

表 12-7　RJ45 接头定义

针　脚	信　号	含　义	输入/输出
1	CAN_H	CAN 接口	CAN 电平
2	CAN_L	CAN 接口	CAN 电平
3	CAN_0V	接地 CAN	—
4 ~ 8	保留	保留	—

12.2　过程数据对象和服务数据对象通信模式

通常我们在 CANopen 总线的通信方式上采用两种方式，即过程数据对象（Process Data Object，PDO）方式和服务数据对象（Service Data Object，SDO）方式。PDO 方式是实时的，SDO 方式是请求的。采用 PDO 通信模式是把指令和数据的传输事先组态好，然后和控制器的输入及输出单元形成映射，这时只要观察控制器的输入及控制输出单元即可完成对设备的

实时监控。指令和数据的传输与总线扫描同步，因此控制和观察设备都是实时的。由于 PDO 交换的数据有限，所以控制什么和观察什么参数要从对象字典中选出，组态在发送 PDO 和接收 PDO 中。一个设备通常有 4 组发送 PDO 和 4 组接收 PDO。一组 PDO 中可以组态 4 个字的指令字或数据字，这样一个设备与控制器之间就可以有 32 个字的数据交换。当控制器上连接的设备越多，并且都配置满了 PDO，则总线上的数据交换就越忙，有时甚至导致某个设备总线传输故障。这就相当于高速公路上，涌入了非常多的车辆，有的甚至是空车，最终导致了交通堵塞。那么解决拥堵的一个办法是提高车速，也就是提高 CANopen 总线的速率，例如从 500K 提高到 1M。另一种办法就是减少空车在路上的行驶，按需进行传输。这就是 SDO 通信模式。SDO 是一种请求式通信，通常我们采用 CANopen 读写功能块，查出设备的索引（Index）和子索引（Subindex），然后执行功能块，发出读写设备请求。

12.3　CANopen 总线的组态

首先我们采用 PDO 方式进行控制器和设备之间的通信，那么我们需要怎样的组态呢？让我们以 LMC058 控制器来做案例加以解读。打开 SoMachine 编程软件，在 LMC058 设备栏目下，会看到有两个 CANopen 总线通信口，CAN0 和 CAN1，如图 12-9 所示。这和我们上面提到的硬件接口是相吻合的。双击 CAN0 或 CAN1 你会看到右边的 CAN 总线信息。在这里你可以选择 CANopen 总线的波特率。点击信息栏目，你可以了解控制器信息。

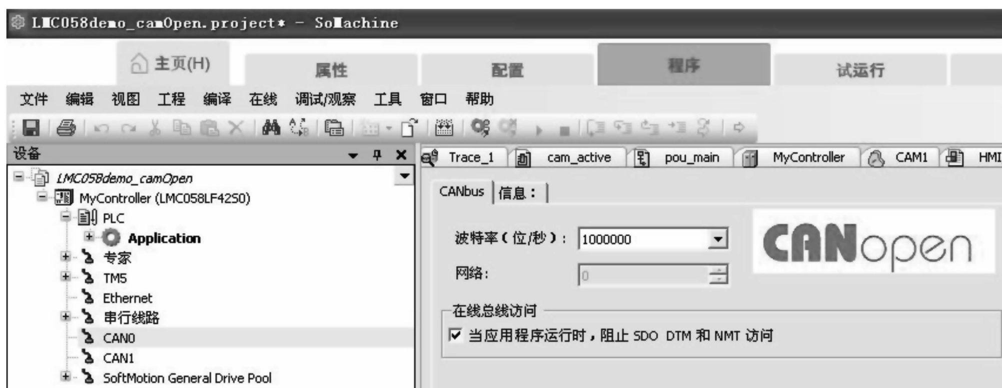


图 12-9　控制器中的 CAN 总线界面

用鼠标选中 CAN 总线某一个接口，例如 CAN0，点鼠标右键，弹出界面操作菜单，如图 12-10 所示。

点击菜单中的“添加设备”，会出现一个添加设备对话框，如图 12-11 所示。在名称栏，你可以填写上要添加的设备名称，比如 lift。在供应商栏目，你可以通过下拉菜单，来选择设备供应商。选择好供应商，下面就会自动显示出提供的总线管理文件及其最新版本号。如果要使用旧的版本，则可以激活显示所有版本。

配置好添加设备对话框后，点击“添加设备”，则这个总线管理界面就被加入到 CAN 总线界面下，同时你会看到在添加设备对话框中的管理文件也没有了，如图 12-12 所示。



图 12-10　在总线界面上添加设备



图 12-11　添加设备对话框



图 12-12　添加总线管理界面

点击设备栏目下总线 CAN0 下挂的总线管理器 lift，你会看到右边添加设备对话框变为总线管理器下包含的所有带有 CANopen 总线的设备，如图 12-13 所示。在这个列表中选择你要连接的设备。例如：连接一个伺服驱动 lexium32A。

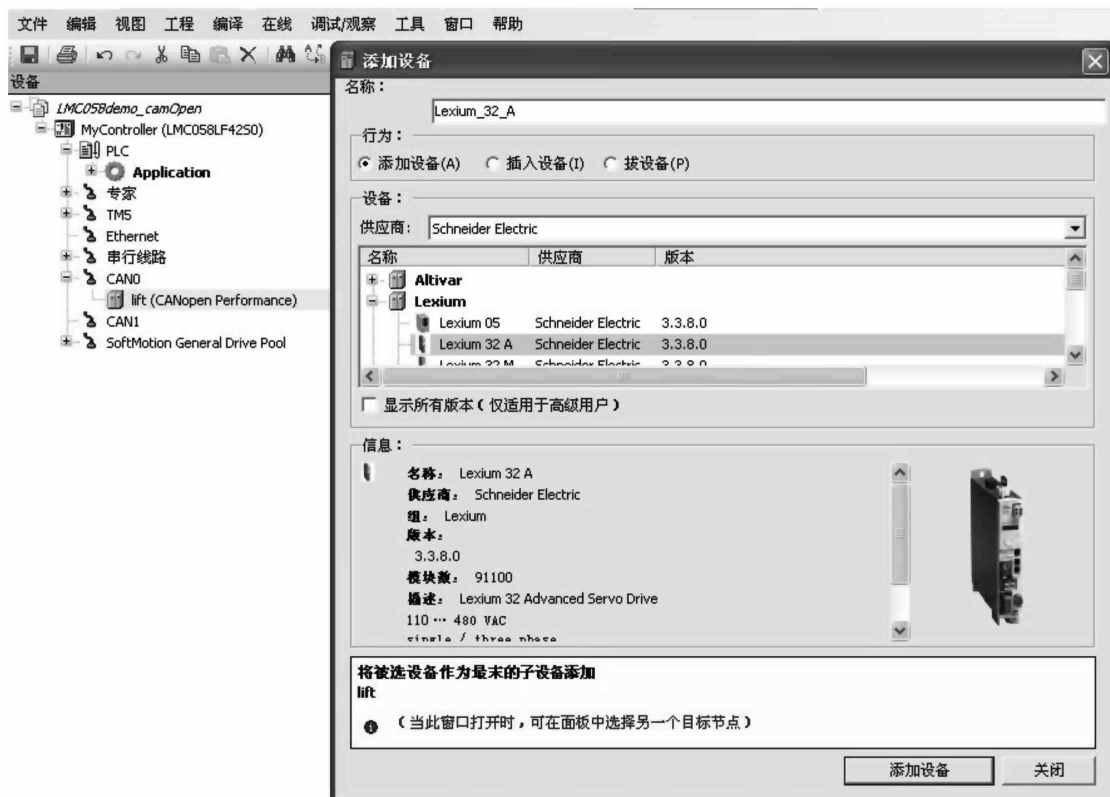


图 12-13　在总线上添加设备

点击“添加设备”，则这个设备就被添加到总线上，如图 12-14 所示。



图 12-14　添加一个设备 lexium32A

当然，我们可以继续从添加设备框中再选择设备添加在这个总线口下，如图 12-15 所示。我们又添加了几个设备。这些设备都连接在 CAN0 口上，总线通信为 CANopen。

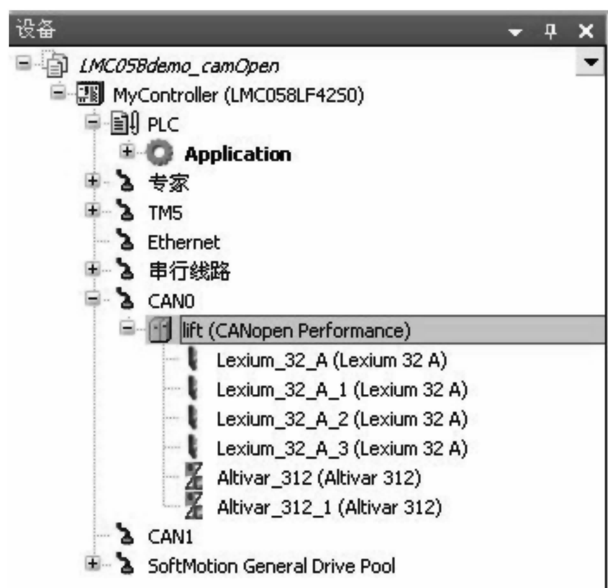


图 12-15　添加多个设备

在硬件配置完毕后，我们就可以对添加的设备进行组态，这里就包括把设备参数与控制器存储单元的映射关系建立起来，从而实现对设备的操控。

双击 CAN0 打开界面，如图 12-10 所示。在这个界面中你可以设置总线主站的通信速

率，一旦设定，连接在 CAN0 上的从站设备都要设定为这个速率。然后双击如图 12-15 中的总线接口界面 lift，打开对总线界面的组态，如图 12-16 所示。



图 12-16　CAN0 总线组态

在总线管理器栏目，主站地址默认为 127。如果激活使能同步生产，则会生成一个总线的同步信号，如 CAN0_Sync。总线数据变化可以用这个信号的周期刷新。如果激活使能 heartbeat 生产，则在此节点会有通信时间的保护。即通信硬件连接的掉线必须在保护时间内恢复，不然的话就要报警。点击 CANopen I/O 映射，打开如图 12-17 所示界面。在这个界面中，你可以看到对总线循环任务的选择。

总线周期选项：该设置选项将在读输入/输出之前和之后对有循环调用的设备有效。它允许设置一个设备总线。

周期任务：默认情况下，“父” 总线循环被设置为有效（使用父总线循环设置），这样将在设备树中搜索下一个有效总线循环任务。

从选择列表中选择你所想要指派的总线任务。该列表提供了当前在应用程序的任务设置中定义的任务。

例如，如果我们选择了总线循环 MAST，那么涉及这个总线数据的程序都必须放在任务设置中的 MAST 里。接下来，我们双击总线接口下的驱动器 d5，打开对 d5 的组态界面，如图 12-18 所示。在这个界面可以对设备的站地址、PDO 映射参数进行定义。也可以观察总线工作状态和映射数据的变化。

在 CANopen 远程设备项目下，我们可以定义这个设备的站地址，这个地址与设备本体的地址要一致。比如，这个站地址为 “1”，那么 d5 的 CANopen 地址也要设为 “1”。

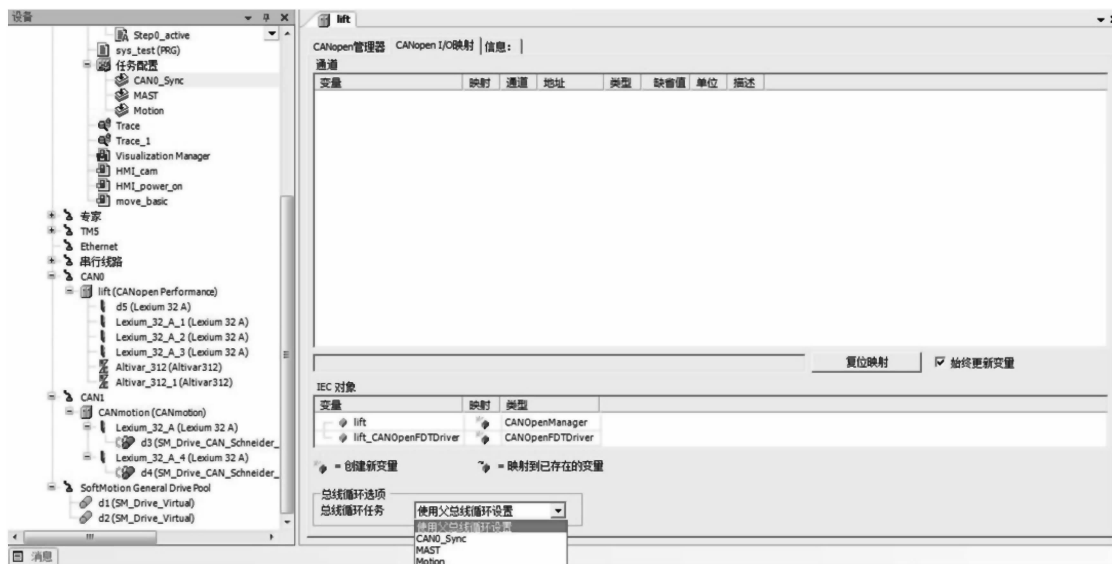


图 12-17　CANopen I/O 映射



图 12-18　总线设备的组态

当激活使能专家设置，栏目中会多出接收 PDO 映射和发送 PDO 映射，如图 12-19 所示。这意味着专家设置是可以对发送和接收的 PDO 进行编辑组态的。

点击“PDO 映射”，得到如图 12-20 所示界面。在这个界面中，我们可以选择要交换的数据。即把驱动器要映射到控制器的参数单元选出。

比如在图中，我们选择了要接收来自控制器对驱动器 d5 的命令参数：

RPDO2：控制字，目标位置；RPDO3：控制字，目标速度。

驱动器 d5 发送到控制器的状态参数：

TPDO2：状态字，实际位置值；TPDO3：状态字，实际速度值。

选择好后，我们就可以点击 CANopenI/O 映射来查看驱动器和控制器之间相互对应的单元，如图 12-21 所示。

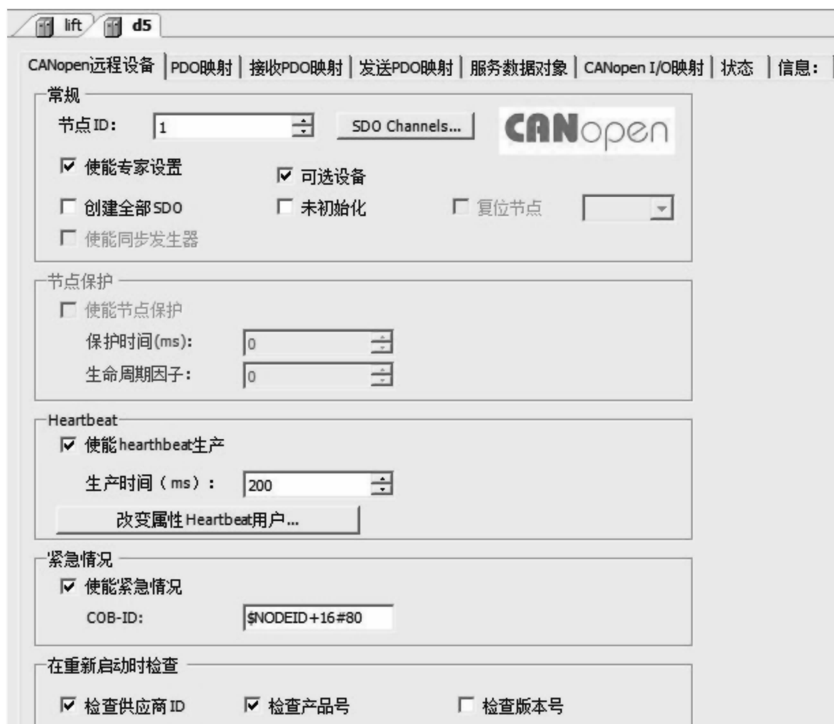


图 12-19　激活专家设置的设备组态

lift　d5

CANopen远程设备 | PDO映射 | 接收PDO映射 | 发送PDO映射 | 服务数据对象 | CANopen I/O映射 | 状态 | 信息：

选择接收PDO (RPDO)

名称	索引	子索引	位长度
☐ **1st receive PDO**	**16#1400**		
driveModeCtrl	16#301B	16#1F	16
RefA16	16#301B	16#22	16
RefB32	16#301B	16#21	32
☑ **2nd receive PDO**	**16#1401**		
Controlword	16#6040	16#00	16
Targetposition	16#607A	16#00	32
☑ **3rd receive PDO**	**16#1402**		
Controlword	16#6040	16#00	16
Target velocity	16#60FF	16#00	32
☐ **4th receive PDO**	**16#1403**		

选择发送PDO (TPDO)

名称	索引	子索引	位长度
☐ **1st transmit PDO**	**16#1800**		
driveStat	16#301B	16#25	16
mfStat	16#301B	16#26	16
motionStat	16#301B	16#27	16
driveInput	16#301B	16#28	16
☑ **2nd transmit PDO**	**16#1801**		
Statusword	16#6041	16#00	16
Position actual value	16#6064	16#00	32
☑ **3rd transmit PDO**	**16#1802**		
Statusword	16#6041	16#00	16
Velocity actual value	16#606C	16#00	32
☐ **4th transmit PDO**	**16#1803**		

图 12-20　PDO 映射选择

在这里，我们看到控制字映射到控制器的输出单元% QW2，% QW6。目标位置设定映射到% QD2，目标速度映射到% QD4。也就是说当我们要使能驱动器，就可以把控制字赋值给% QW2 和% QW6 。而位置设定值和速度设定值的给定，我们就可以通过对% QD2 和% QD4 的赋值发送到驱动器里。在这个图中，我们实际上选上了两个控制字，多占了控制器

的一个单元。那么当激活了专家模式后，栏目中多出了接收 PDO 和发送 PDO，这样我们就可以在这两个栏目中编辑 PDO 了。点击“接收 PDO 映射”，如图 12-22 所示。

lift | d5

CANopen远程设备 | PDO映射 | 接收PDO映射 | 发送PDO映射 | 服务数据对象 | CANopen I/O映射 | 状态 | 信息：

通道

变量	映射	通道	地址	类型	缺省值	单位	描述
		Controlword	%QW2	UINT			
		Targetposition	%QD2	DINT			
		Controlword	%QW6	UINT			
		Target velocity	%QD4	DINT			
		Statusword	%IW6	UINT			
		Position actual value	%ID4	DINT			
		Statusword	%IW10	UINT			
		Velocity actual value	%ID6	DINT			

图 12-21　驱动器参数与控制器单元的映射

lift | d5

CANopen远程设备 | PDO映射 | 接收PDO映射 | 发送PDO映射 | 服务数据对象 | CANopen I/O映射 | 状态 | 信息：

名称	索引	子索引	位长度
1st receive PDO communication	16#1400	16#00	
driveModeCtrl	16#301B	16#1F	16
RefA16	16#301B	16#22	16
RefB32	16#301B	16#21	32
2nd receive PDO communication	16#1401	16#00	
Controlword	16#6040	16#00	16
Targetposition	16#607A	16#00	32
3rd receive PDO communication	16#1402	16#00	
Controlword	16#6040	16#00	16
Target velocity	16#60FF	16#00	32
4th receive PDO communication	16#1403	16#00	

添加PDO... 添加映射... 删除... 编辑...

图 12-22　对 PDO 进行编辑

选中图 12-22 中的 PDO1，按“删除”键，选中 PDO3 的控制字，按“删除”键，这样就简化为如下 PDO 条目，如图 12-23 所示。

同理，点击“发送 PDO 映射”，如图 12-24 所示。

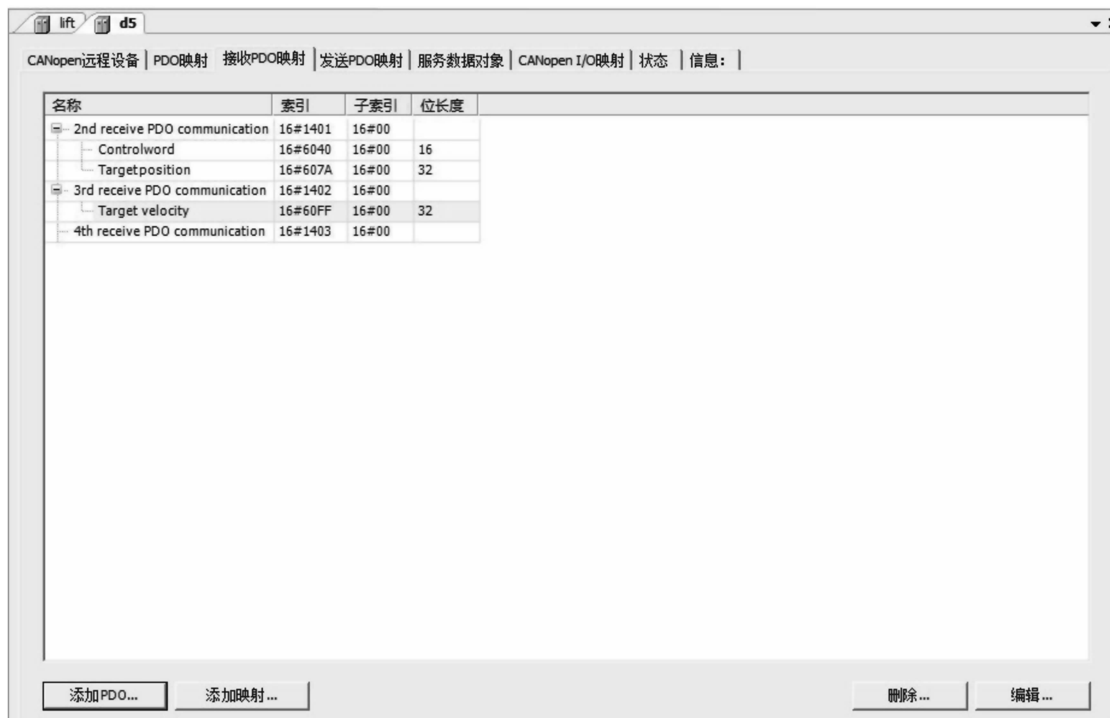


图 12-23　删除多余条目后的接收 PDO

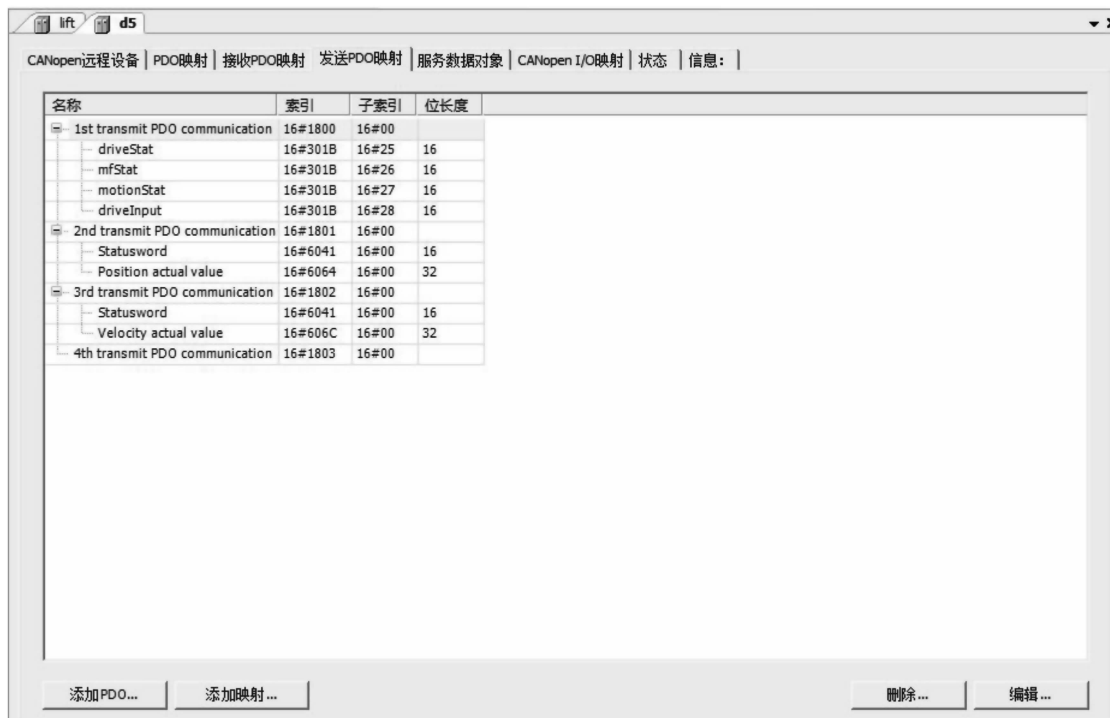


图 12-24　发送 PDO 编辑

选中 PDO1 按“删除”键，选中 PDO3 状态字，按“删除”键，最后编辑的 PDO 如图 12-25 所示。

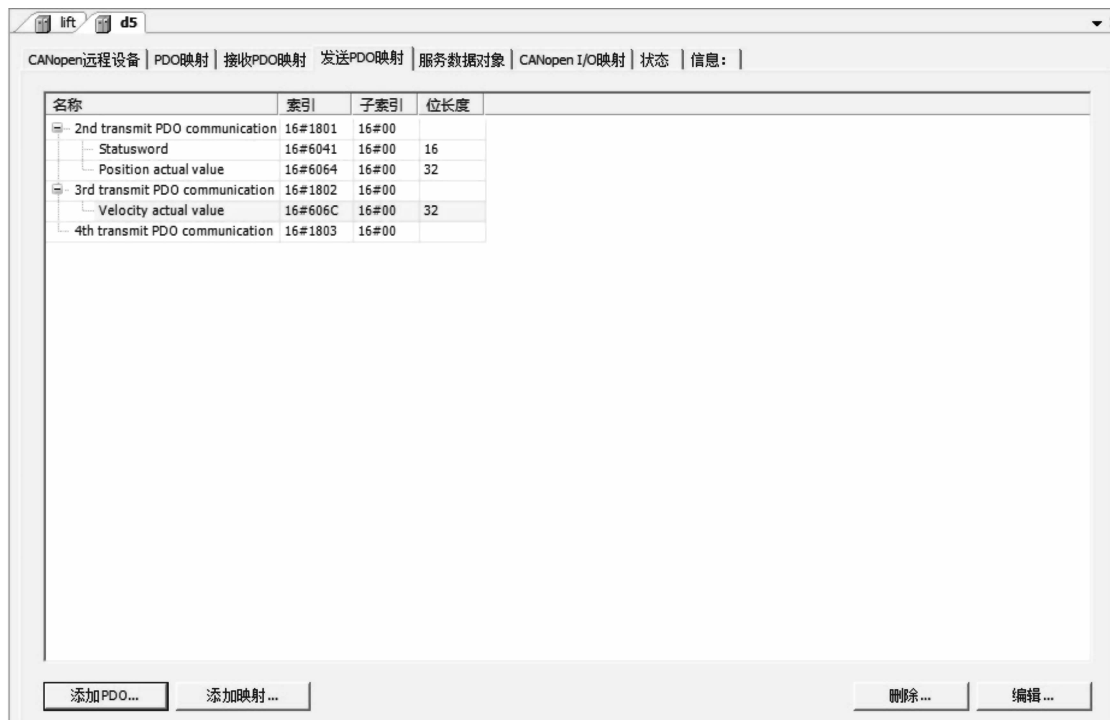


图 12-25　编辑后的发送 PDO

点击“CANopen I/O 映射”，我们看到的对应映射如图 12-26 所示。通过对 PDO 的编辑大大消减了对控制器资源的占用，使得控制更加清晰、简洁。



图 12-26　编辑后的 PDO 映射

这样当电动机转动时，我们就可通过% ID4 知道电动机运动到的位置，通过% ID5 就可知道实际的运动速度。而对电动机的控制，我们可以查找驱动器手册关于控制字的指令码来顺序赋值给% QW2 即可。

在有些要求高速通信的情况下，我们会发现虽然选择了 1M 通信速率，但还是感觉数据的交换不是很及时。这是因为组态默认的通信都是异步的。所以对接收 PDO 和发送 PDO 而言，我们还可以定义它的数据交换方式。在如图 12-25 所示的发送 PDO，我们点击“编辑”键，打开 PDO 属性，如图 12-27 所示。在 PDO 属性中，我们看到默认的传输类型为异步传送，即当数据发生变化或其他某一事件发生后才传送数据，这会产生一定的延时。



图 12-27　PDO 属性

为了实时刷新数据并传送，我们可以选择同步传送，如图 12-28 所示。



图 12-28　定义传输类型

这样，我们就可加快数据的交换。而同步的周期可以在图 12-16 所示组态界面中定义。

当然，在添加总线设备上，我们也可以选中 CAN1，点击鼠标右键，选择“添加设备”，而这时的添加设备对话框显示出两种 CAN 总线界面：一种是我们已遇到过的 CANopen 总线；另一种是 CANmotion，也就是我们说的可以做电子齿轮、电子凸轮和 CNC 的同步运动总线。也就是说，CAN1 的总线配置可以有两种选择，如果不做多轴的同步，可以选择 CANopen Performance，如果在这个总线上的轴要做同步控制，则需要选择 CANmotion，如图 12-29 所示。

运动总线下的伺服驱动和轴参数描述，如图 12-30 所示。我们看到连接在同步运动总线下的伺服驱动，除了对驱动器的描述栏 Lexium_32_A_4 外，还多了对电动机轴参数的描述栏，如 d3 和 d4。



图 12-29　选择同步运动总线

双击 d3，打开电动机轴参数的描述，如图 12-31 所示。

在这个框的基本单元中，我们可以设置轴的运动模式。可以设置为虚轴模式，也可以设置为旋转轴或无限轴模式（模数），设置为无限轴模式后，就需要设置位置自动清零的模数，即走多少用户单位后，位置自动清零。例如，电动机转一圈 360°，如果模数设为 360，则电动机向前运行时的位置数总是在 0～360 之间周期递增变化。当选择有限轴时（限定的），就会看到有软件限位的设置。为了减小轴的冲量，运动方式可以设为梯形曲线运动和 S 型曲线运动。点击缩放/映射，就会看到如图 12-32 所示界面。在比例缩放的第一行是电动机转一圈对应编码器的分辨率。如 d3 电动机转一圈，编码器增量 131072（十六进制 20000）。这个数值是由电动机编码器决定的，所以是固定的。第二行是电动机轴因带减速或增速装置而产生的输出位置比例。例如电动机轴带了一个 15：2 的减速机，也就是说电动机旋转了 15 圈，齿轮箱输出轴转动 2 圈。则第二行左边填入 15，右边填入 2。当比例出现小数时，左右可同时扩大 10 的整数倍。第三行是把齿轮输出的旋转单位转化为用户单位。例如，齿轮输出轴转动 2 圈，对应位置变化 100mm。则第三行左边填 2，右边填 100，这里用户单位是毫米。



图 12-30　运动总线下的伺服驱动和轴参数描述

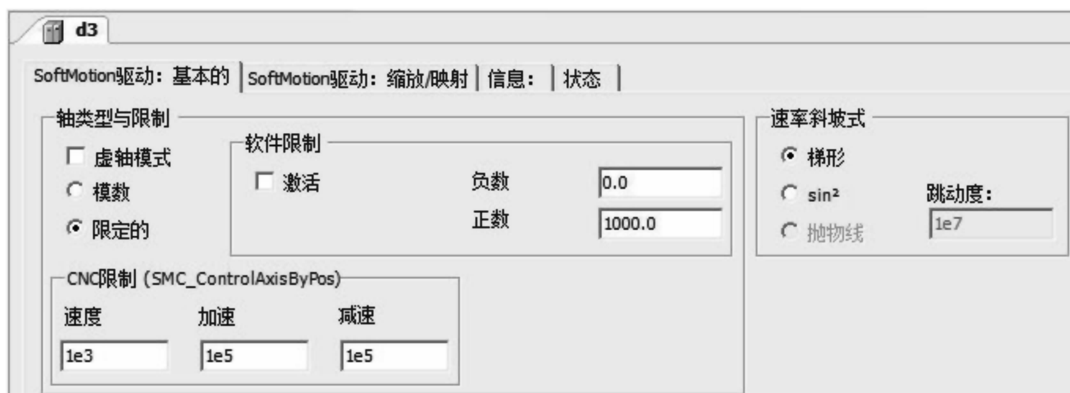


图 12-31　轴参数描述框

在映射功能区，你可以看到驱动器参数可以自动映射到控制器上的内部单元。如图中所示，驱动器状态字映射到 LMC058 运动控制器的“%IW30”，实际位置映射到控制器的“%ID16”单元。这些参数的自动映射，我们可以在驱动器的描述栏进行配置，这同上述 CAN0 的配置一样。



图 12-32　比例缩放和映射

12.4　直接请求的数据交换 SDO

采用 PDO 方式做数据交换，简单、直接。但受到数量限制，而且用不用这些数据都会时时占用总线，会造成连接在总线上的设备数量不能太多。当需要的数据不是实时需求，而是随机需求时，则可以采用 SDO 方式。按这种方式的需要，直接发出数据交换请求。在编程时，我们可以采用 CAN 总线读写功能块来实现对驱动器参数的读出和对驱动器设置参数的写入。例如，在程序中调入一个运算块，打开输入助手，点开 CIA405 下拉功能，就会看到 CANopen 总线的读写集群，如图 12-33 所示。

在这个集群中，有 4 个功能块：SDO_READ，SDO_WRITE 这一组对参数的读出/写入参数对象是任意大小的；另外一组 SDO_READ4，SDO_WRITE4 是读/写对象的 4 个字节。通常来说，使用比较多的是读/写对象的 4 个字节。在输入助手中，我们选择 SDO_ READ4 并确定，调入此功能块如图 12-34 所示。

在这个功能块中 NETWORK 是 CAN 总线口通道。

1（默认值）为第一个 CAN 总线口，如 CAN0。

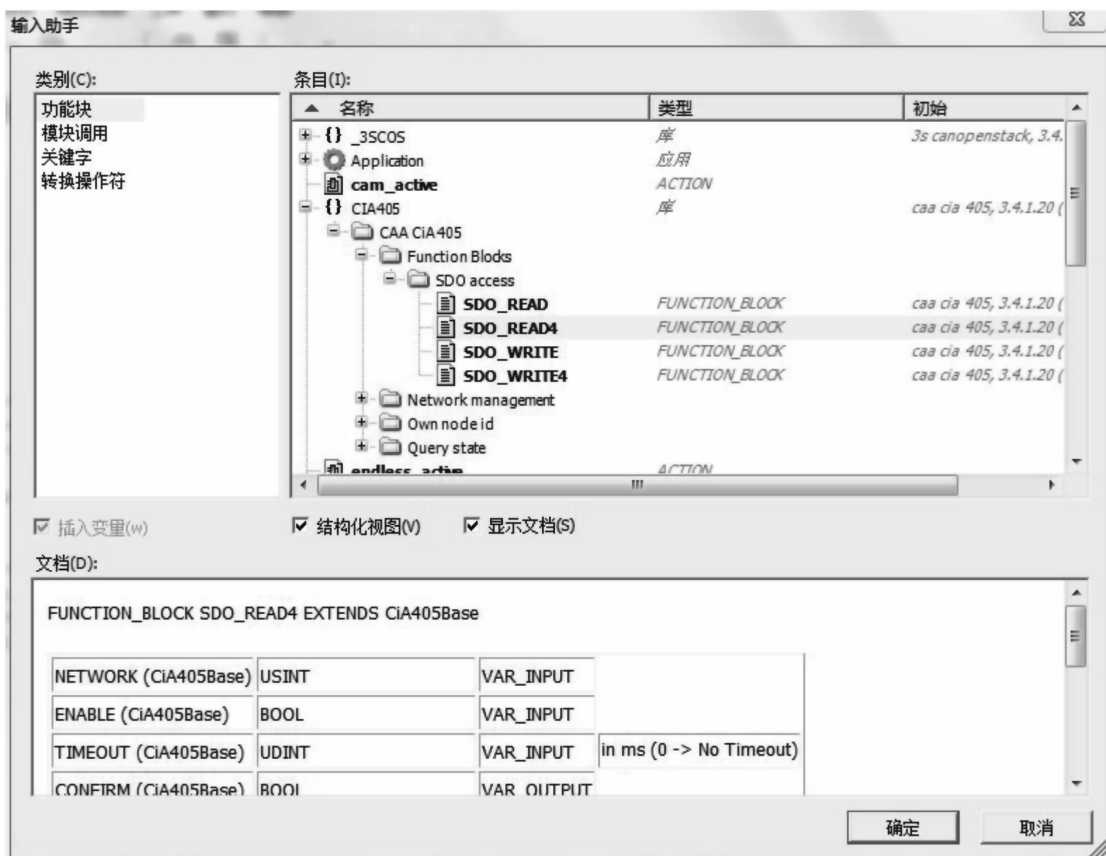


图 12-33　CANopen 总线读写集群

2 为第二个 CAN 总线口（如果存在），如 CAN1。

ENABLE：启用功能块的执行。

在上升沿：执行开始。

在下降沿：如果执行未完成，则取消执行。否则，输出数据会复位为 0。

TIMEOUT：最大执行时间（以毫秒为单位）。如果在收到响应之前达到超时，则执行会因超时错误而中止。

0（出厂设置）＝禁用超时。

1...65535＝超时值(以毫秒为单位)。

DEVICE：读取设备的站地址。

CHANNEL：SDO 通道号，默认值为 1。

INDEX：对象索引。

SUBINDEX：对象子索引。

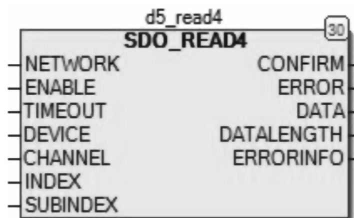


图 12-34　CANopen 总线的读取

所以，根据引脚定义，当我们要读出驱动器的参数，只需查出参数的对象索引和子索引，填上驱动器的地址就可以了。例如，我们查看一下 d5 的状态。根据图 12-24 所示我们知道对象索引是 16#6041，子索引是 0。因此，编写的读状态程序如图 12-35 所示。当 k8 为“真”，执行读取，读取的

参数由 DATA 输出，DATA 是一个数组，有 4 个字节，如图 12-36 所示。

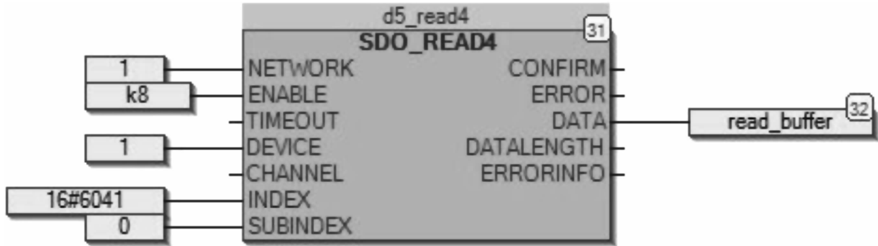


图 12-35　读取驱动器状态程序

Read_Buffer	ARRAY [1..4] OF BYTE	
Read_Buffer[1]	BYTE	16#04
Read_Buffer[2]	BYTE	16#00
Read_Buffer[3]	BYTE	16#00
Read_Buffer[4]	BYTE	16#00

图 12-36　读出的数据

我们还可以通过 CAN 总线对驱动器执行写入命令，这时我们采用写功能块，如图 12-37 所示。

在写功能块中 DATA 是 4 个字节，所以命令字是 16#0000000F，查一下驱动器手册中的驱动器命令字如图 12-38 可知，命令字的字长是两个字节，所以 DATALENGTH 填“2”。

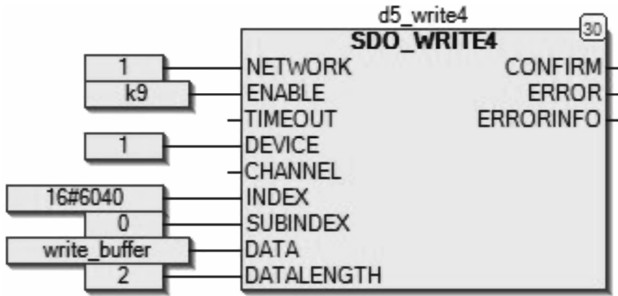


图 12-37　对驱动器写入使能命令

需要注意的是：如果字长填的不对，写命令就会报错。如果查不到这个字长或有些手册标注的字长有错误，那么可以通过读命令的办法获得字长。

在 SoMachine 编程软件中，施耐德电气针对自己的伺服驱动器 LEXIUM32A，也有一组读写功能块，如图 12-39 所示。

DCOMcontrol	DriveCom 控制字码 Bit 编码参见操作、运行状态一章 Bit 0：Switch on Bit 1：Enable Voltage Bit 2：Quick Stop Bit 3：Enable Operation Bit 4...6：由运行模式决定。 Bit 7：Fault Reset Bit 8：Halt Bit 9：Change on setpoint Bit 10...15：已保留（必须为 0） 变更的设置将被立即采用。	- - - -	UINT16 UINT16 读 / 写 - -	CANopen 6040:0$_h$ Modbus 6914

图 12-38　驱动器命令字

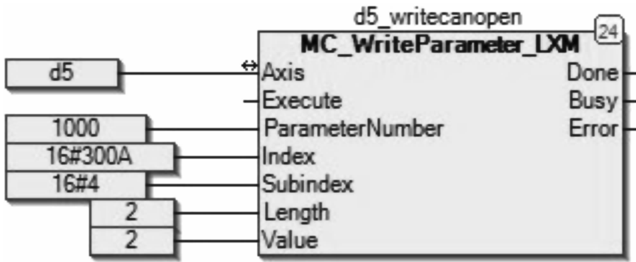


图 12-39　LEXIUM32A 驱动器的写功能块

当 CAN1 组态为 CANmotion 后，连接在这个总线口的伺服驱动器要采用 softmotion 功能块。

CANmotion 总线口的读写如图 12-40 所示。

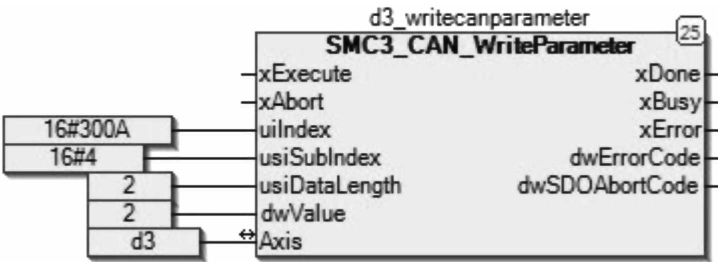


图 12-40　CANmotion 总线口的读写

有了这个句型，想读出或写入伺服驱动某些功能和参数，就进行设备的索引和子索引的替换即可。

第 13 章　总线设备工具及设备类型管理器应用

企业自动化需要两种主要的数据流：一种是“垂直”数据流，从企业层向下到现场设备，包括信号和组态数据；还有一种是“水平”通信，发生在现场设备之间，使用同样的或者不同的通信技术进行传送。

随着现场总线集成到控制系统中，还需要完成一些其他的任务。除了现场总线和设备指定工具，需要一个把这些数据集成到更高层系统的工程工具。特殊情况，为了使控制系统扩展范围和异类连接，典型的在流程工业领域，工程接口的清晰定义是非常重要的，这使用户可以方便地使用所有设备。

一些不同的制造商指定了必要的使用工具。例如，施耐德电气对变频器的调试采用 So-move 调试软件，对伺服驱动的调试采用 CT 调试软件。从系统生命周期管理和工厂层面自动化的观点来看，这些工具中的数据通常是不可见的数据。

为了确保在一个工厂范围内或一台机器的控制系统中，对控制和自动化设备管理的一致性，有必要对现场总线、设备和子系统进行全面集成，使覆盖整个自动化生命周期中的所有自动化任务成为一个无缝的整体。

IEC 62453 提供了一个接口规范，用于开发总线设备工具（Field Device Tool，FDT）组件，在一个客户机/服务器架构中，支持功能控制和数据访问。这个标准接口的可用性使得多个制造商开发服务器和客户机应用非常便利，并且支持开放的互操作。

一个设备或者模块特定的软件组件，被称为一个设备类型管理器（Device Type Manager，DTM），是由某个制造商提供的、具有相关设备类型或者实体类型的信息。每个 DTM 通过定义的 FDT 接口可以集成到工程工具中。这种集成方法通常对所有的现场总线开放，因此可以把不同设备和软件模块集成到控制系统中。

IEC 62453 公共应用接口对有兴趣的应用开发者、系统集成商、现场设备和网络设备的制造商提供了支持。它还增加了采购选择，减少了系统成本和提供了对生命周期的管理帮助。明显节省了控制系统在运行、工程和维护上的费用。

IEC 62453 系列标准支持：

1）用于异类现场总线环境、多个制造商设备、功能块和模块化子系统的通用工厂层面工具，实现所有自动化领域（比如：流程自动化、工厂自动化和类似监视与控制应用）生命周期管理；

2）在控制系统中，包括现场总线、设备、功能块和模块子系统之间，实现综合和一致的生命周期数据交换；

3）简单和强大的制造商独立性特征，可以把不同自动化设备、功能块和模块化子系统集成到一个控制系统的生命周期管理工具之中。

13.1　FDT 及 DTM 概述

DTM 设备类型管理器：设备驱动软件，由设备制造商提供。用于进入设备组态数据，修改参数，并且会提供图形化用户界面，以方便用户操作。它会被集成在控制器软件中。如图 13-1 所示。

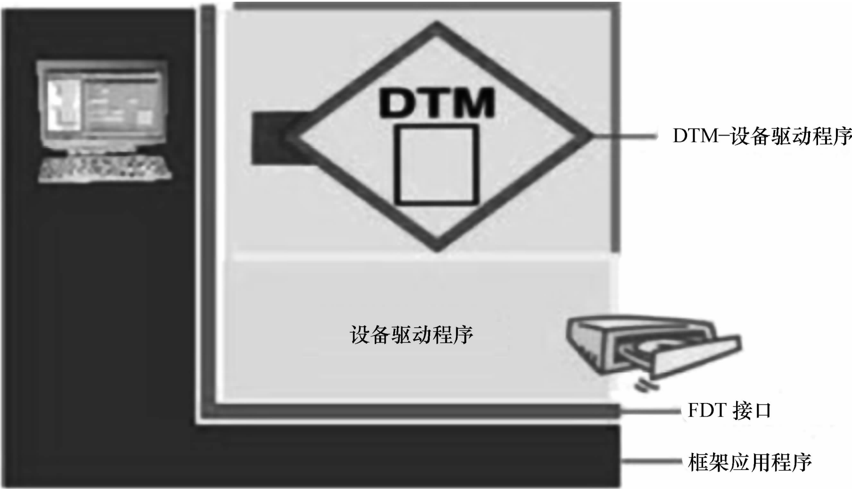


图 13-1　DTM 设备驱动软件

FDT 现场设备工具：连接在总线设备和控制器软件（SoMachine）之间的数据交换界面。它能把总线设备和控制器软件连接起来，实现交互。它包含了公共环境、设备类型管理、数据管理、网络组态和导航，功能结构如图 13-2 所示。

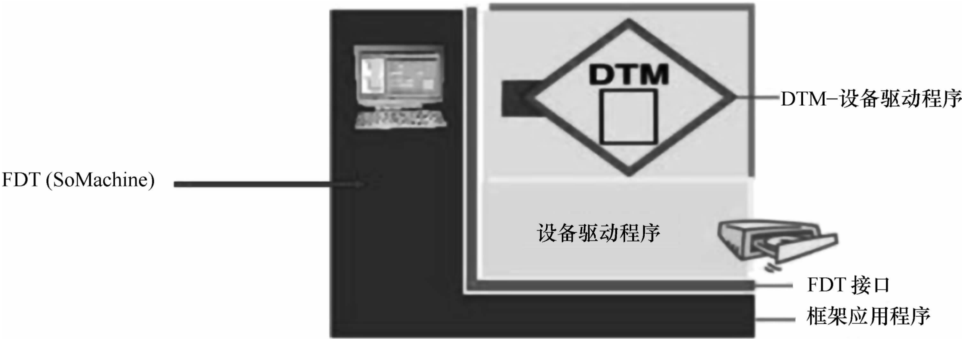


图 13-2　FDT 功能结构

综上所述，我们知道 FDT/DTM 概念的提出是基于很多设备在应用中需要对数据进行组态；在线修改参数；使用调试工具进行调试等需求。例如，我们在调试伺服驱动器时，需要单独一个调试软件 Somove，对 PI 参数进行调整，或对 I/O 点进行功能设置等，我们希望这些工具能集成在控制器中，当我们对这些设备进行参数修改，配置变更时不用再启动工具软件，而直接使用控制器软件就可完成。那么 FDT/DTM 就帮助我们做到了这点。

首先，FDT/DTM 都是通过标准通信界面，采用一种控制器软件来操控挂在任何总线上的设备，如图 13-3 所示。

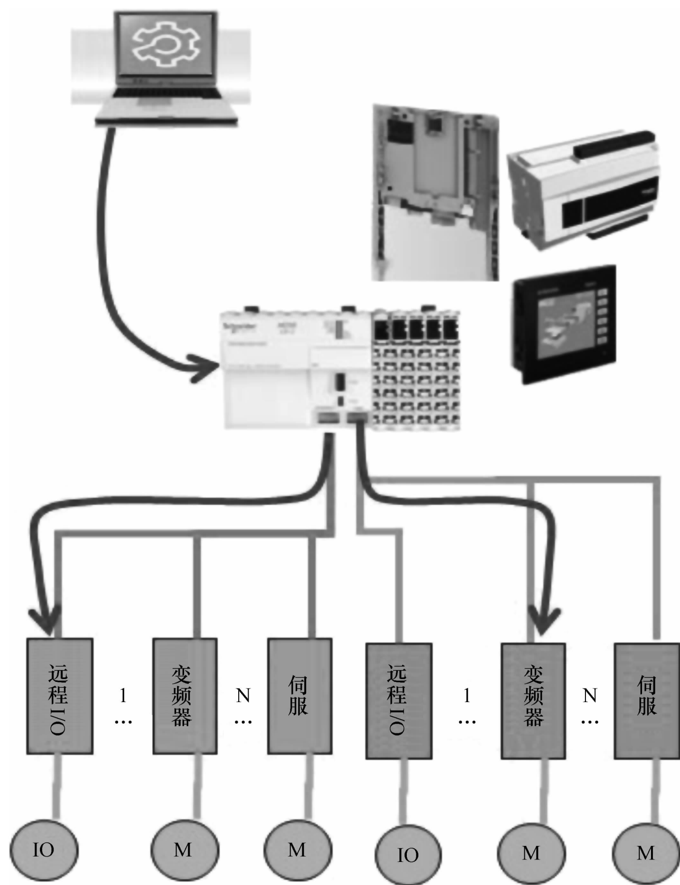


图 13-3　FDT/DTM 对设备的操控

任何总线设备由于有了 FDT/DTM 的帮助，能够实现只要把设备挂在控制器上，就能够即挂即用，设备的组态、参数改变都能够通过 DTM 的用户界面加以操作，大大方便了设备的组态和调试。

13.2　DTM 的组态

挂在总线上的每一个设备的 DTM 驱动软件由设备生产商提供。因此，首先要把这个软件安装到或复制到计算机中，然后在控制器 SoMachine 编程软件中，使用工具栏目的设备库来添加 DTM 驱动，如图 13-4 所示。

打开设备库后会出现如图 13-5 所示界面。这个界面用于设备库管理和设备驱动安装。

在图 13-5 所示界面中，点击安装 DTM 键，会进入计算机，扫描已经安装的 DTM 文件并显示出来，如图 13-6 所示。

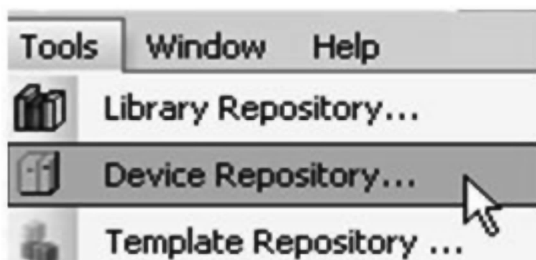


图 13-4　在 SoMachine 编程软件的工具栏下打开设备库

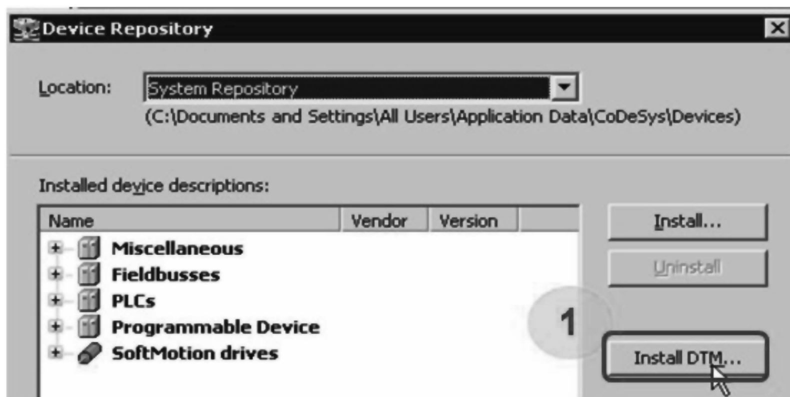


图 13-5　设备库管理界面

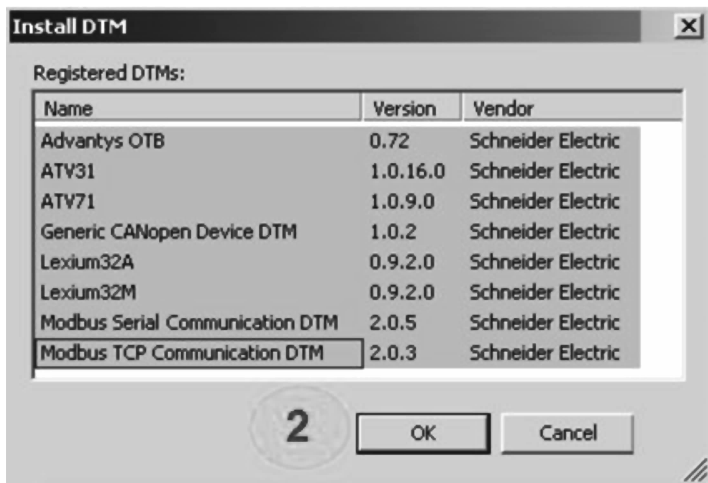


图 13-6　已经安装在计算机中的 DTM 驱动文件

选择要安装的驱动，并点击“OK”确认安装，这样在计算机中的驱动软件就被安装到控制器 SoMachine 编程软件中。安装完后，在设备目录中带有 DTM 的设备被激活。在组态一个控制系统时，我们就可以把带有 DTM 的设备加入到系统中，并且直接使用控制器软件 SoMachine 打开 DTM 界面对设备进行配置或观察。如图 13-7 所示，我们在 CANopen 总线上添加一个设备，点击鼠标右键，打开下拉菜单，选择添加。

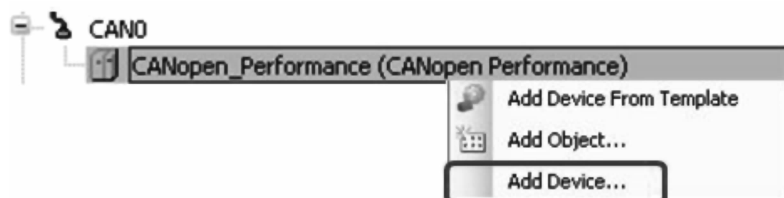


图 13-7　在总线上添加设备

打开设备添加后，会出现设备列表，如图 13-8 所示。例如，添加一台变频器 Altivar71。我们选择带有 DTM 的驱动，并确认。这样被添加的设备便带有 DTM。

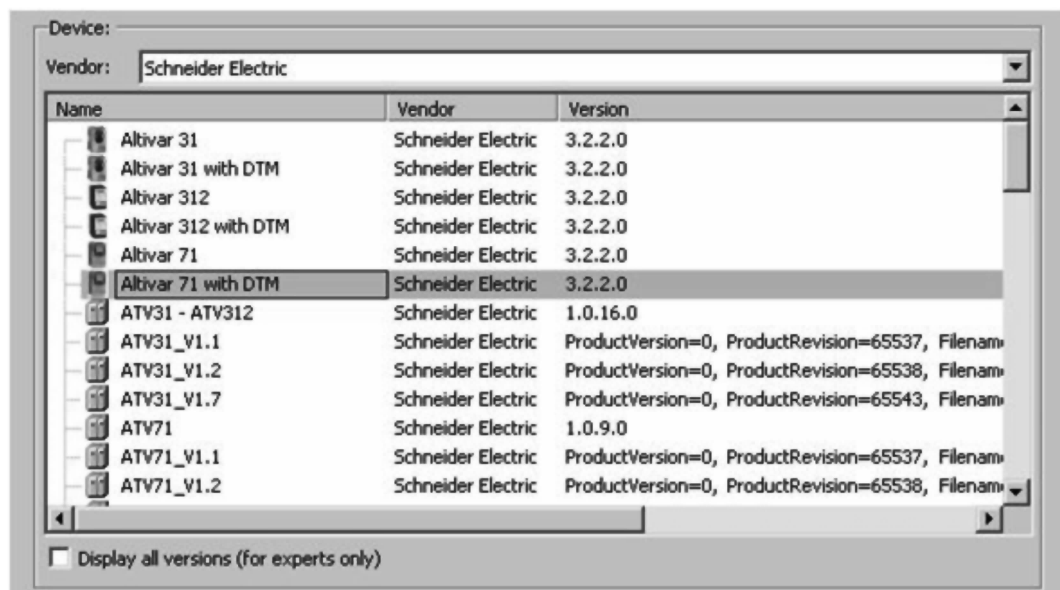


图 13-8　添加带有 DTM 的设备

选中挂在总线上的设备，点鼠标右键打开下拉菜单，如图 13-9 所示。

选择设备组态“Device Configuration”，选择建立关联“Create topology”，如图 13-10 所示。

在建立关联中选择合适的设备，例如一台变频 ATV31，如图 13-11 所示。

建立关联后，要观察设备参数时，可以按菜单流程，选择标准功能“Standard function”，然后点击“Observe”，如图 13-12 所示。

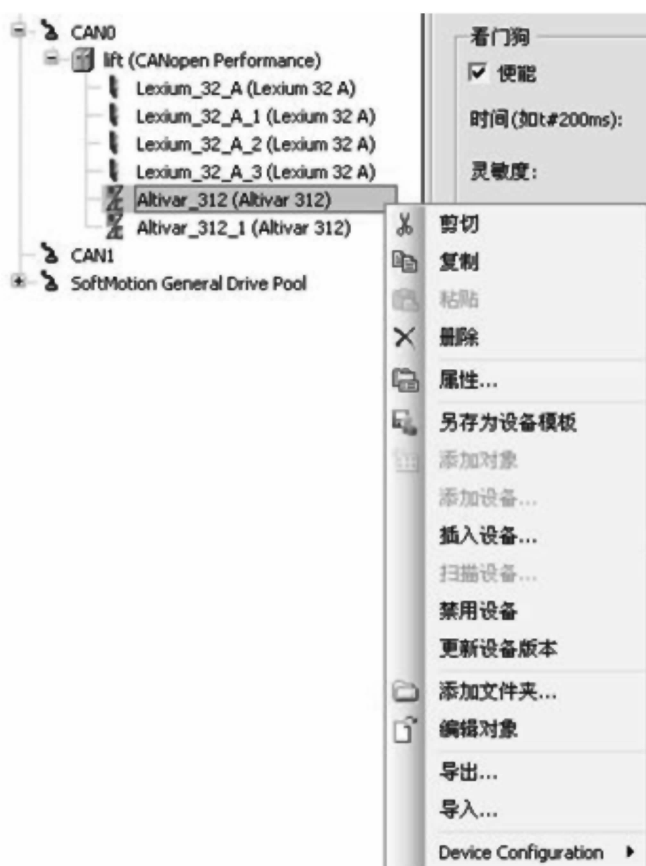


图 13-9　打开设备下拉菜单

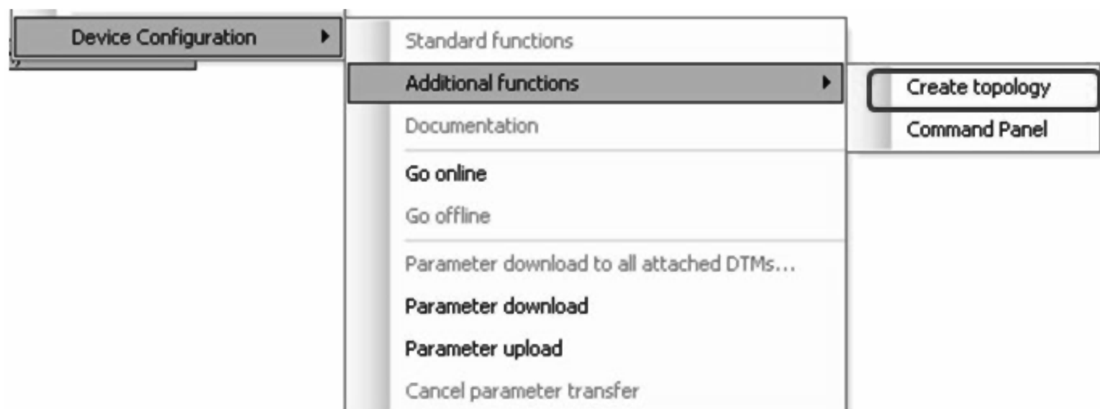


图 13-10　选择建立关联

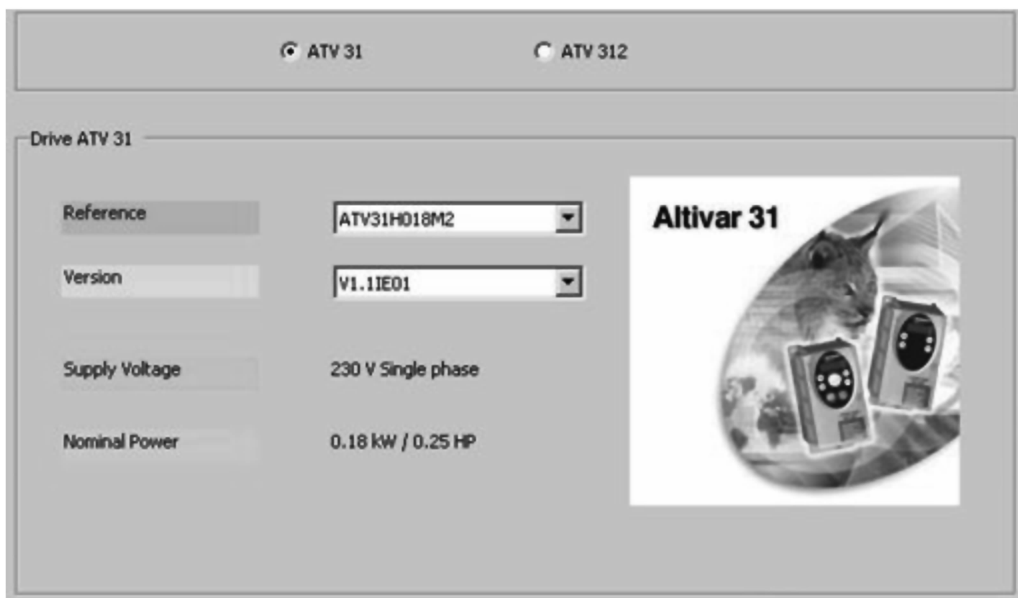


图 13-11　建立关联



图 13-12　选择标准功能

打开图形界面可以操作或观察设备的参数，设备的参数界面如图 13-13 所示。

当需要在线操作一下设备，例如，对电动机进行点动 JOG 操作或强制某个 I/O 点动作等，就需要选择在线“Go online”或建立和设备的在线连接以及对设备参数进行下载或上传，如图 13-14 所示。

在线后还可选择同步监视目前的一些参数值，如图 13-15 所示。

可以看到，这些对设备的操作都已经嵌入到控制器软件 SoMachine 中。它的使用大大方便了使用者对设备的调试、测试和组态，使得设备接入控制器变得“即插即用”。通过下面的案例，大家可以看到带有 DTM 设备的组态和调试是多么的方便。

案例 1：在 CANopen 总线上加入一个驱动器 Lexium32A

如图 13-16 所示，在 CANopen 总线上加入带有 DTM 的驱动器 Lexium32A。注意，选择添加设备时，选择 Lexium32A　Advanced Settings。这时右边工作区会弹出驱动器和电动机的图片供组态参考。

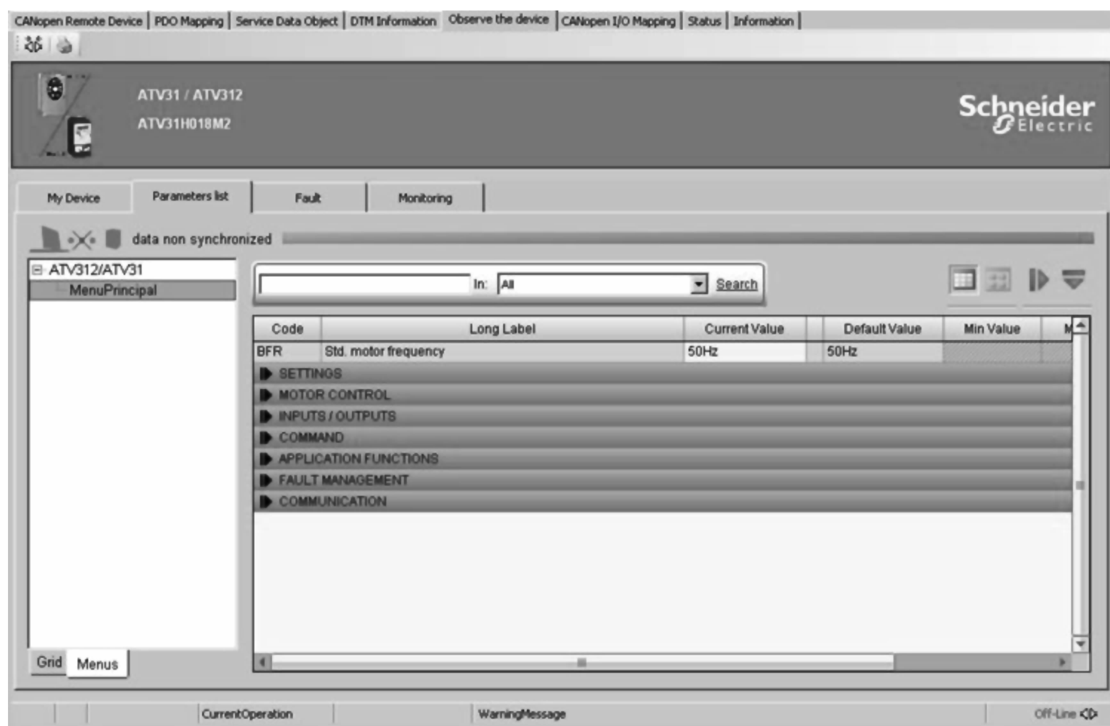


图 13-13　设备的参数界面

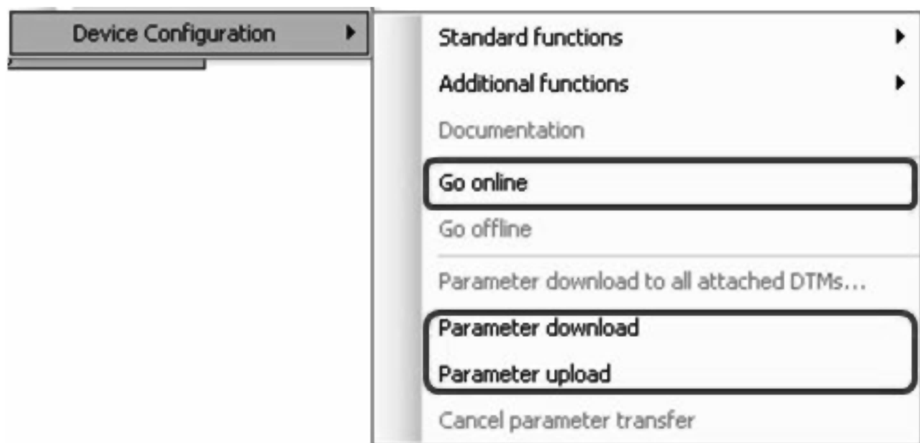


图 13-14　在线操作设备

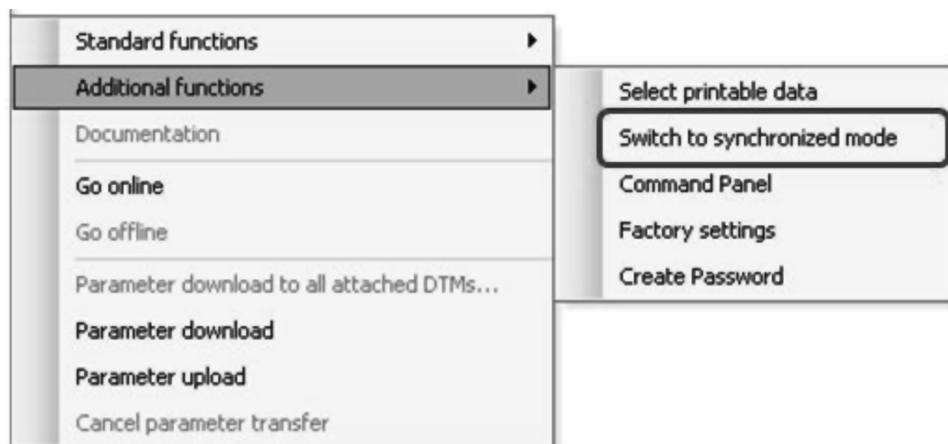


图 13-15　同步模式

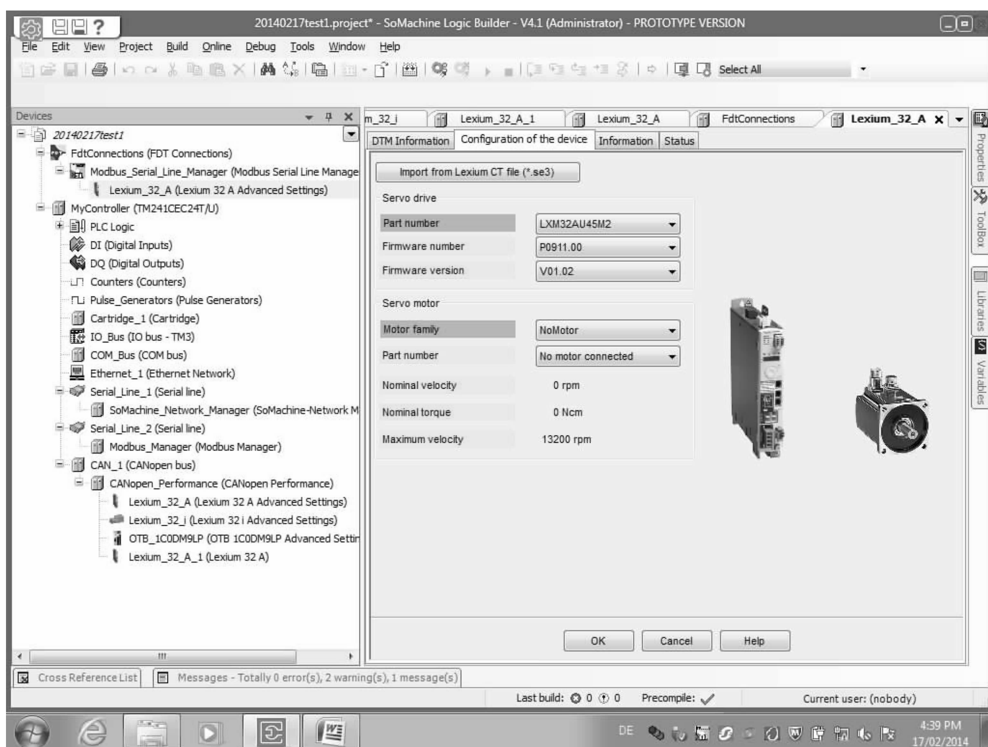


图 13-16　加入一个驱动器 Lexium32A

通过图中设备的下拉菜单，我们可以轻松地选择驱动器和电动机，或通过驱动器调试软件存的驱动器参数文件，把驱动器和电动机的配置导入设备组态界面。当我们选择好驱动器和电动机如图 13-17 所示并确认后，设备的 My Device 栏目会显示出驱动器和电动机的相关参数，如图 13-18 所示。

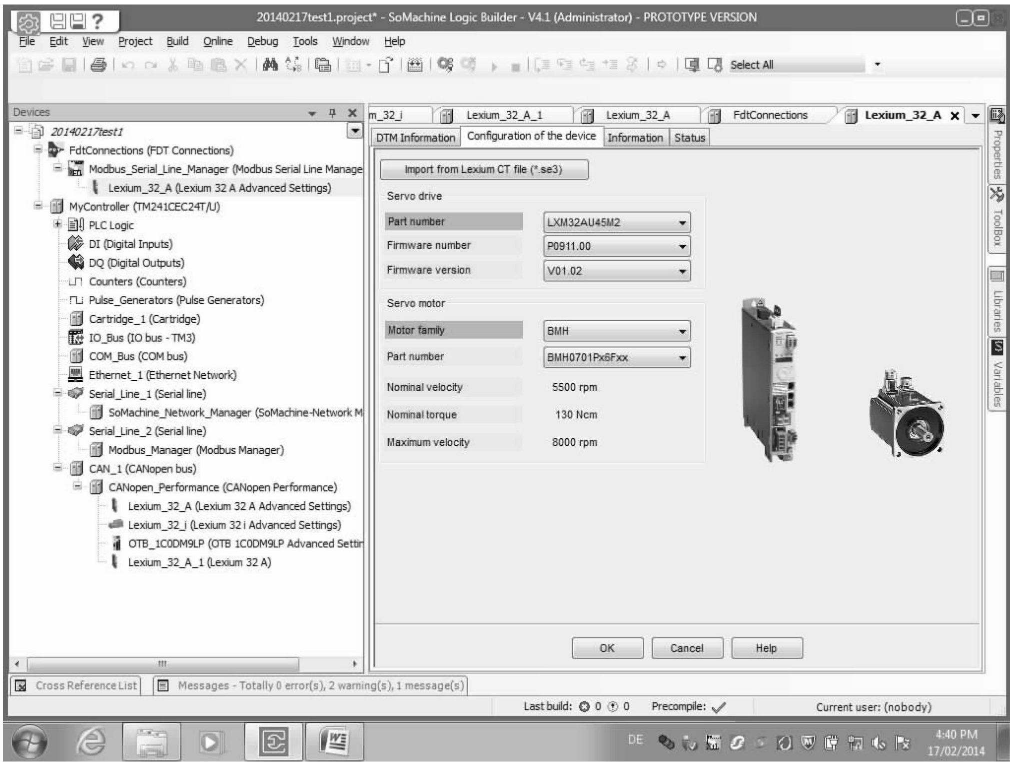


图 13-17　选择驱动器和电动机

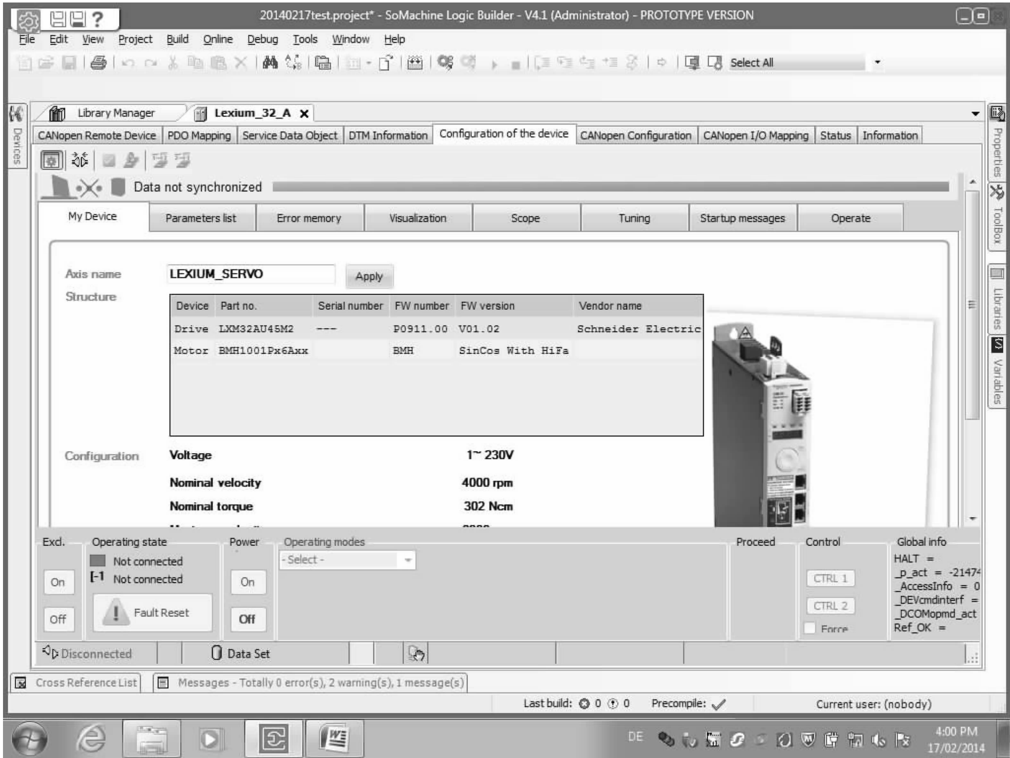


图 13-18　配置好的驱动系统

通过驱动系统界面，我们可以对参数进行修改，对伺服电动机进行操作、整定等。设备的状态和操作界面如图 13-19 所示。

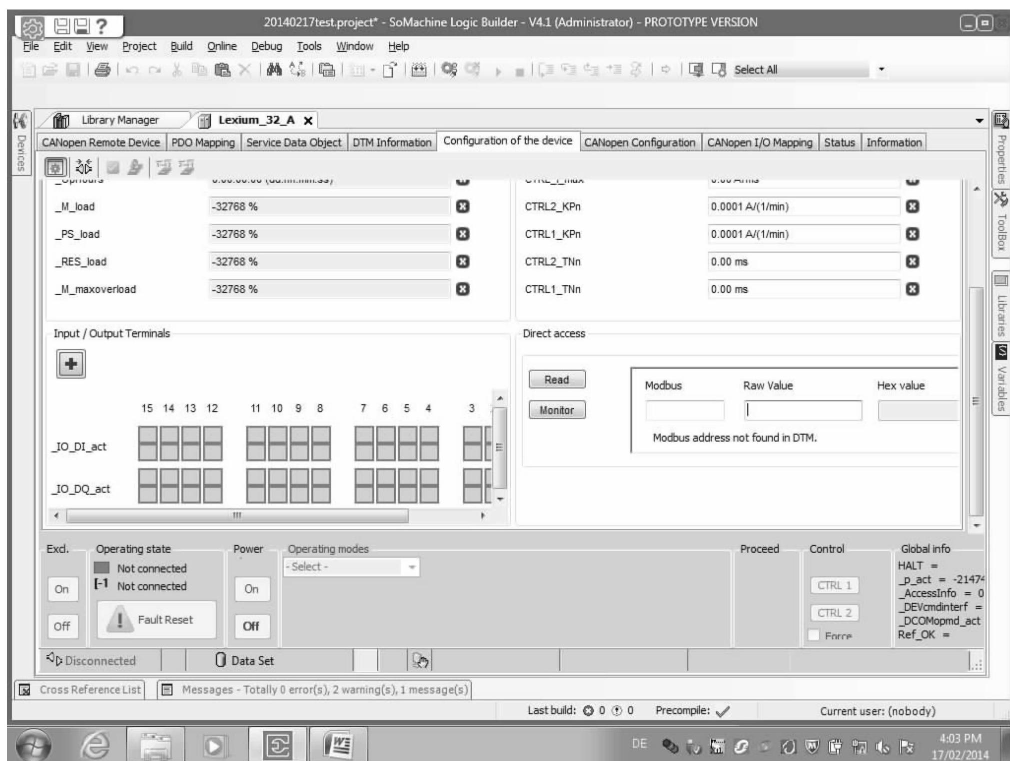


图 13-19　设备的状态和操作界面

通过操作界面直接操作电动机运动。使电动机使能，选择运动模式，观察电动机运动，对电动机进行自整定。这样就可以不用调试软件了。

案例 2：添加远程 I/O OTB 模块

添加设备选择 OTB，如图 13-20 所示。

在此界面上进行组态，点击图中导轨，会出现设备目录，如图 13-21 所示。

组态完后，如图 13-22 所示。

确认后，可对 I/O 进行操作或观察，如图 13-23 所示。

案例 3：添加一个一体化电动机 LEXIUM32i

在 CANopen 总线下添加设备，选择 LEXIUM32i 一体化电动机，如图 13-24 所示。右边工作区显示出设备图片。

通过设备下拉菜单，确认电动机和连接的总线，确认后，电动机的操作界面弹出，如图 13-25 所示。

通过设备界面，对电动机进行操作，如图 13-26 所示。

还可以通过设备界面，对所选定的参数进行观察，如图 13-27 所示。

对设备状态进行观察，如图 13-28 所示。

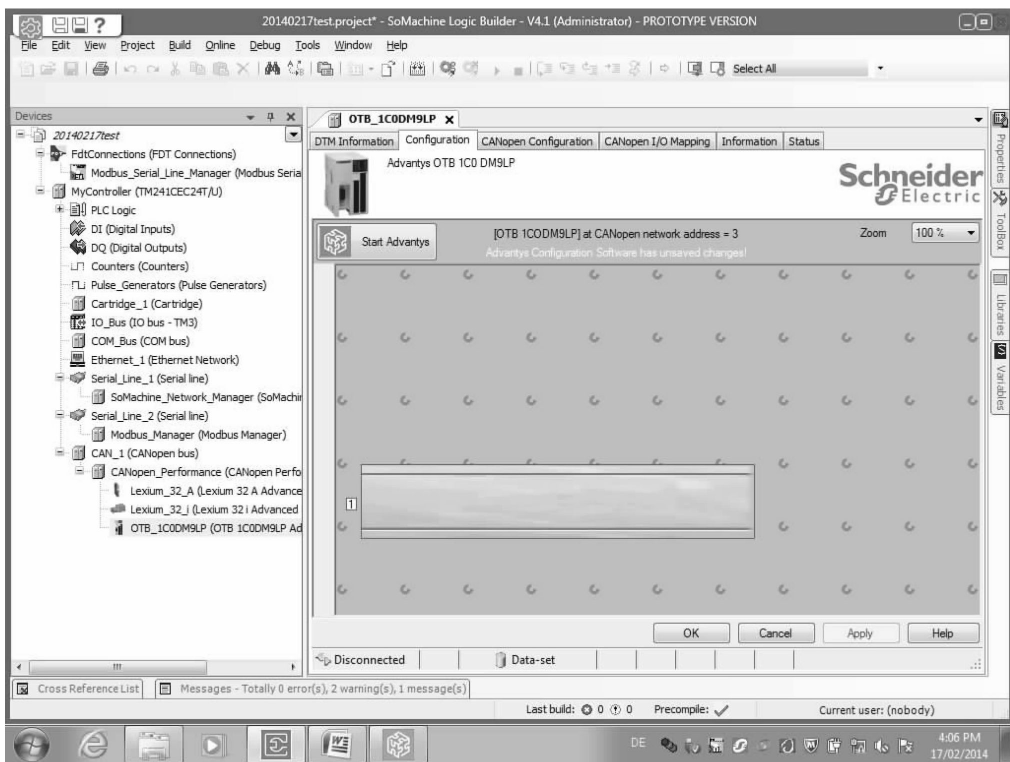


图 13-20　添加一个远程 I/O OTB

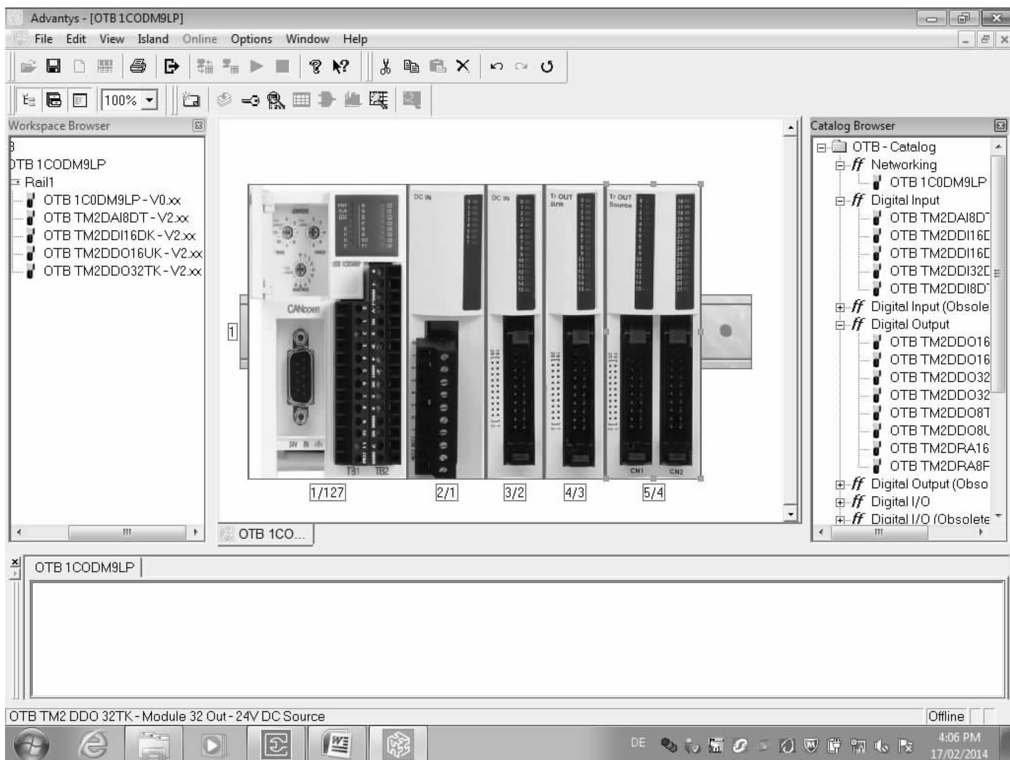


图 13-21　组态远程设备 OTB

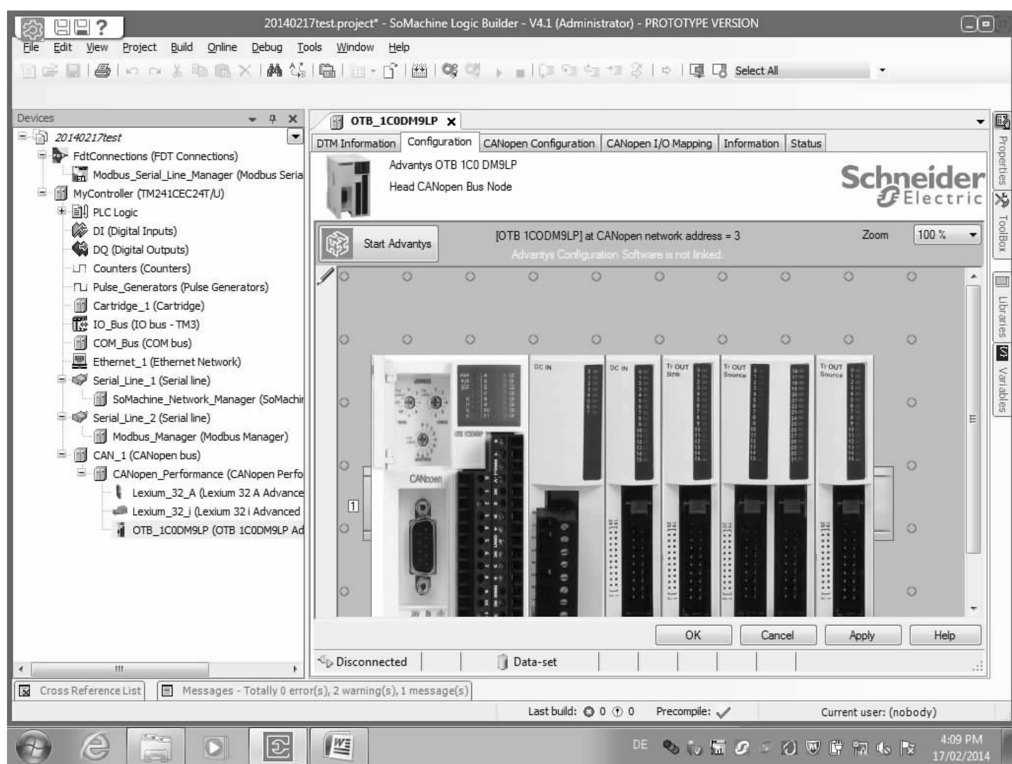


图 13-22　组态好的 OTB

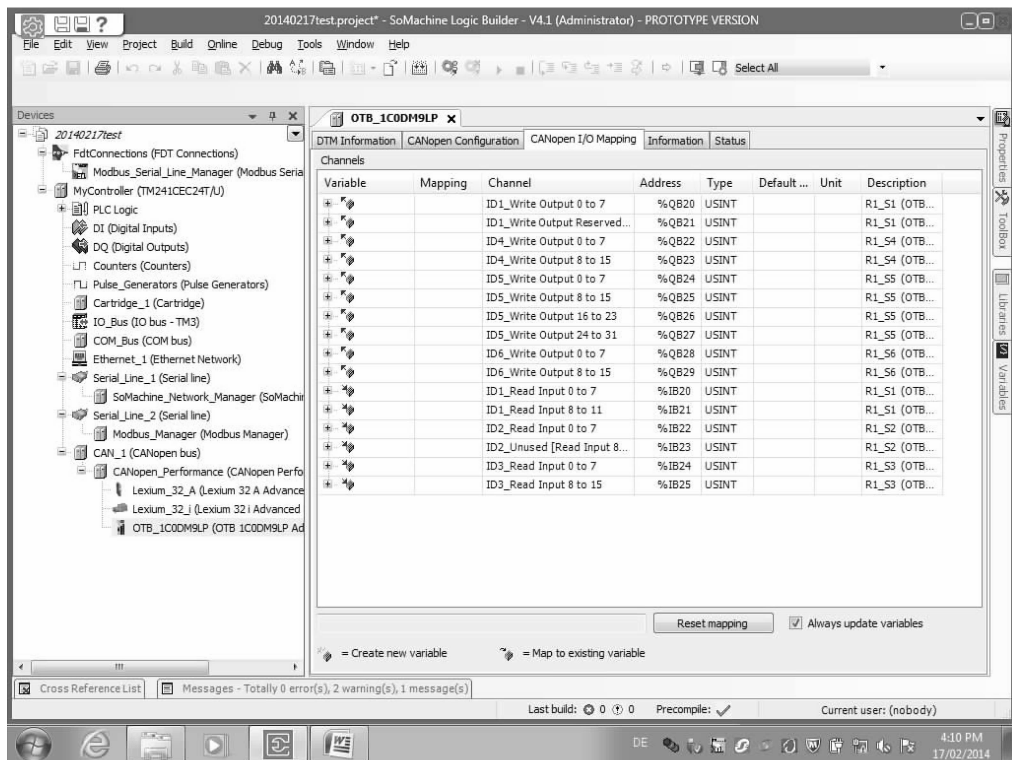


图 13-23　对远程 I/O 进行操作或观察

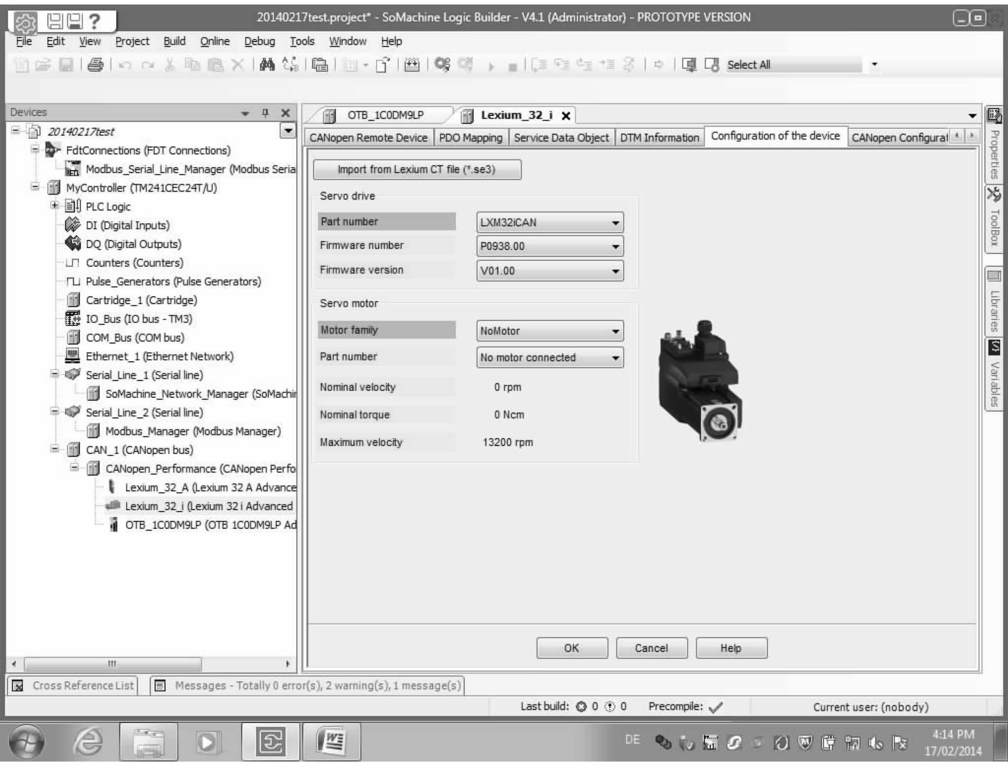


图 13-24　添加一个一体化电动机

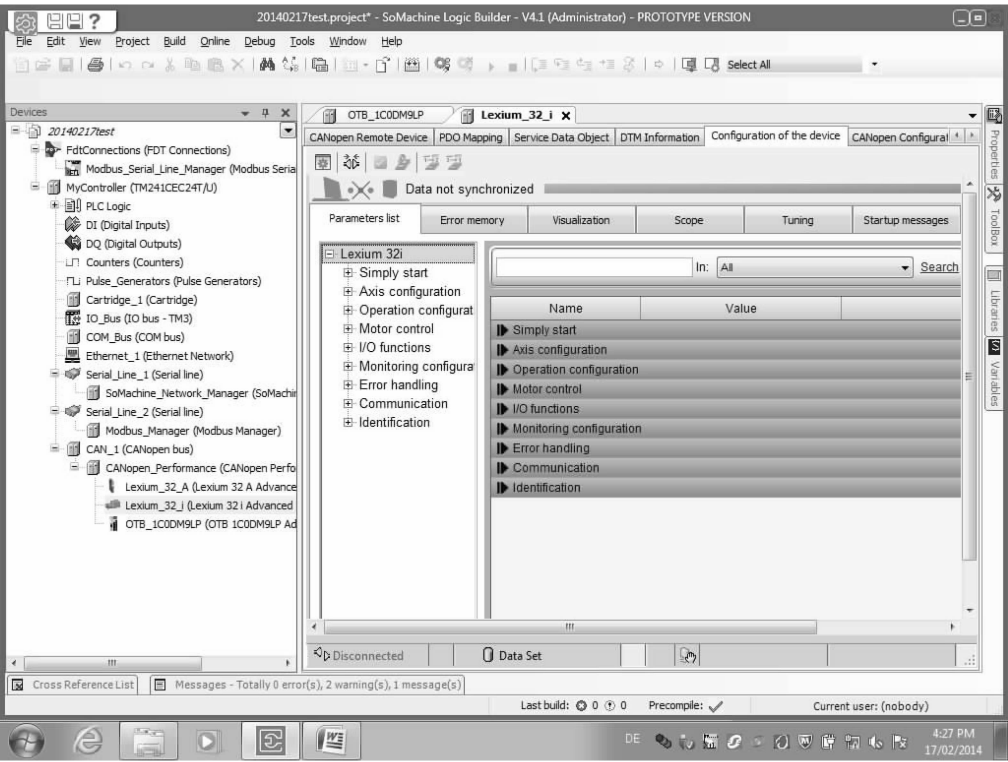


图 13-25　设备的参数和操作界面

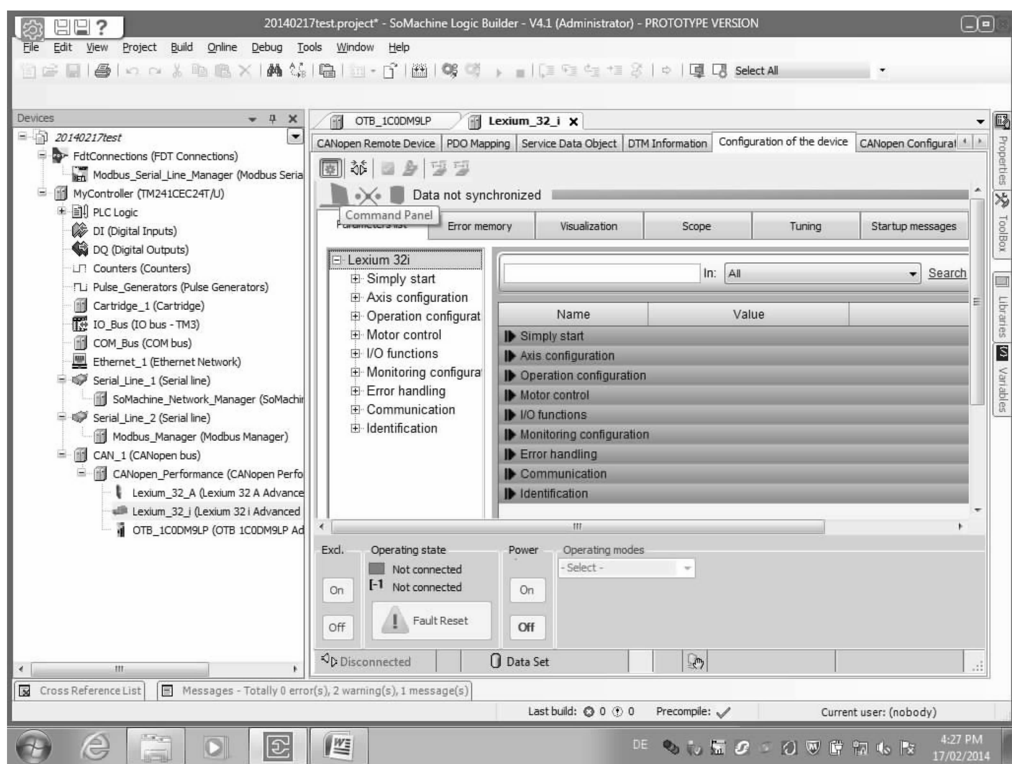


图 13-26　通过设备界面操作电动机

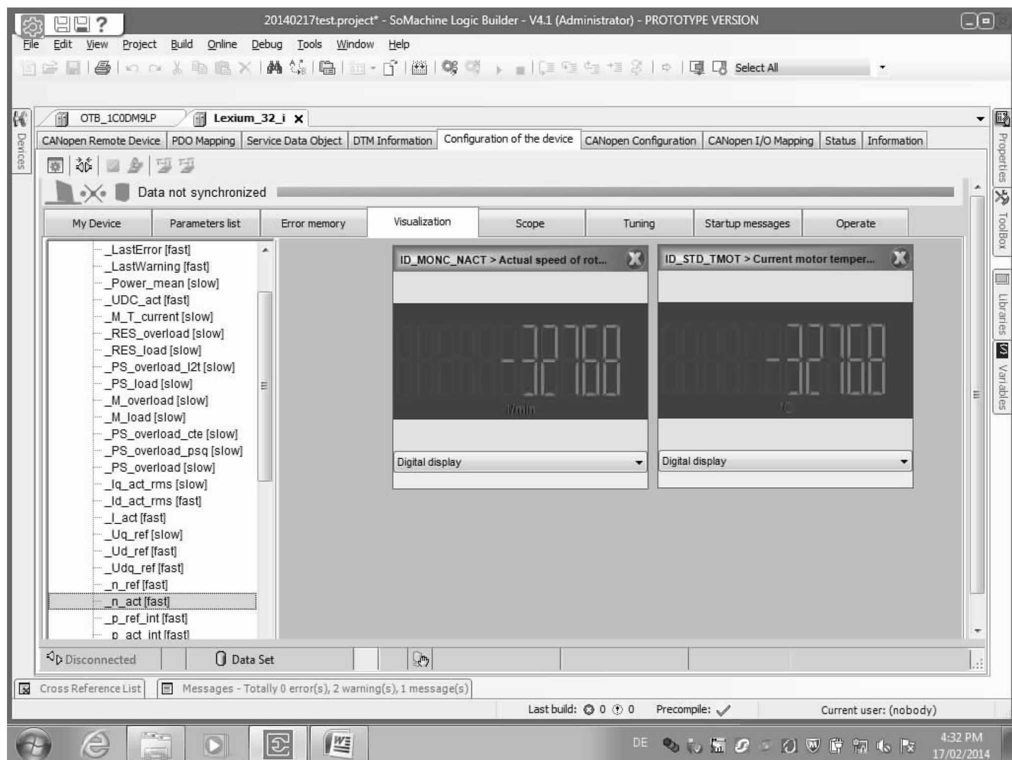


图 13-27　对选定的电动机参数进行观察

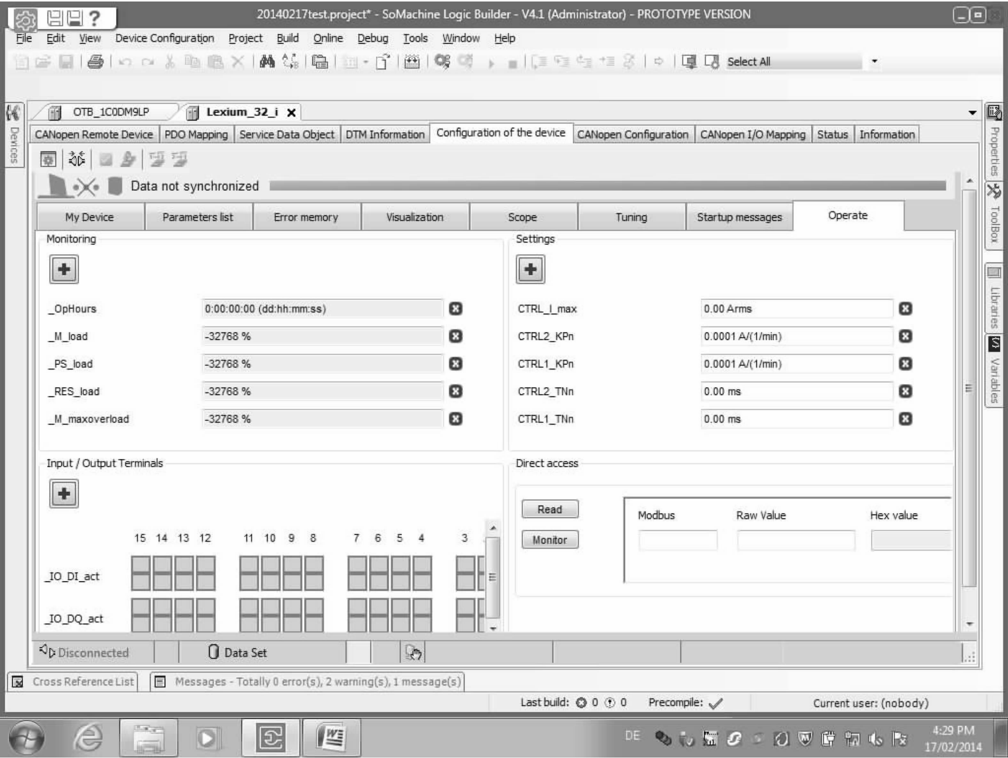


图 13-28　对设备状态进行观察

第 14 章　简单运动控制功能的实现

我们知道，通常实现对伺服电动机的运动控制可以有三种方法：第一种是脉冲和模拟量控制，由于是一路通道对应一个轴的控制，我们也称其为单轴控制；第二种是总线控制，由于一路总线可以控制多台电动机，我们也称其为多轴控制；第三种是驱动器内部运动任务控制，这种控制可以没有上位控制器，驱动器输入点就可以启动事先存好的各个运动任务或运动任务序列，因此我们也称其为独立控制。那么，在 SoMachine 控制平台，我们提供了具有脉冲控制功能的单轴控制以及具有总线控制的多轴控制功能。

14.1　采用脉冲控制的设计方法

在 SoMachine 编程软件控制平台，M238 逻辑控制器提供了 2 路脉冲串输出（Pulse Train Output，PTO）用于运动控制功能。如图 14-1 所示，控制器可以产生不同频率脉冲串和设定的脉冲个数输出来控制伺服和步进电动机的速度和位置。在 SoMachine 软件中包含了简单运动控制功能的功能块，如速度指令和位置指令功能块等。这种 PTO 控制的位置定位功能是一种开环控制方式，用于一些简单机器的定位操作。

在 M238 逻辑控制器本体提供了最多 2 路的脉冲控制。每一路脉冲串输出都是 24V 电平。脉冲输出类型可以被组态为脉冲/方向（P/D）或正脉冲/负脉冲（CW/CCW）。本体的一个辅助输入点可以被组态为驱动器准备好信号（Drive ready）的检测或原点到达信号（Origin）的检测。要注意的是这个点是 24V 高电平有效，当组态为原点信号时，这个点是常闭的，不然，会报警。脉冲输出的频率最高达到 100kHz，如图 14-2 所示。



图 14-1　脉冲控制伺服或步进



图 14-2　脉冲输出最大频率

脉冲输出的两种模式如图 14-3 所示。

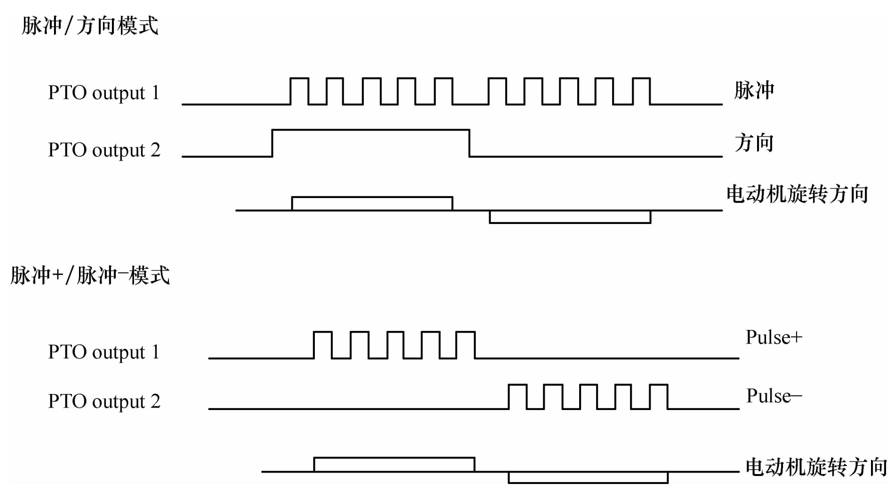


图 14-3　脉冲输出的两种模式

在组态 PTO 通道时，每一路 PTO 都有：

1） 一个辅助输入点。

2）2 个快速输出点。

它们的功能如图 14-4 所示。并且这些点都是固定功能的。

通过PTO预定义I/O	PTO0	PTO1	说明
标准输入点 输入 - Drive ready / Origin	I9	I12	驱动器准备好或原点信号
快速输出点 Out1 Out2	Q0 Q1	Q2 Q3	脉冲串输出 1 方向信号输出 2

图 14-4　运动控制时 I/O 点的功能

因此，我们看到在编辑一个采用脉冲驱动的运动任务之前，先要对控制器的运动功能及相关 I/O 点进行组态。打开带有脉冲输出功能的 M238 逻辑控制器，你会看到在内置功能“Embedded Functions”栏目下，有一组 PTO_PWM 功能区，如图 14-5 所示。

点击这组功能，会打开对脉冲串输出和脉宽调制功能（PTO/PWM）的组态界面，如图 14-6 所示。

在这个界面中我们可以看到有两组脉冲可以组态。点击其中一个 PTO，例如 PTO 0。在 PTO 00 的模式 Mode 一栏中，通过点击值下的下拉菜单我们可以把这组脉冲配置为脉冲串输

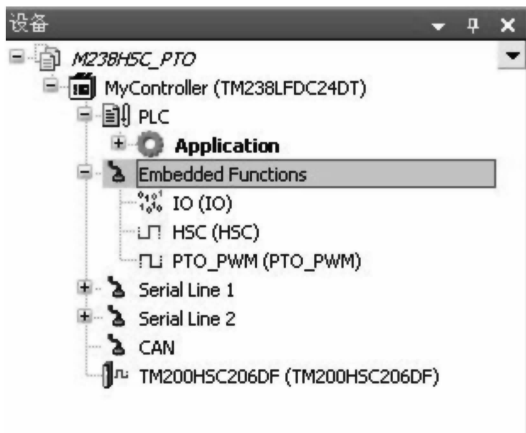


图 14-5　控制器的内置功能

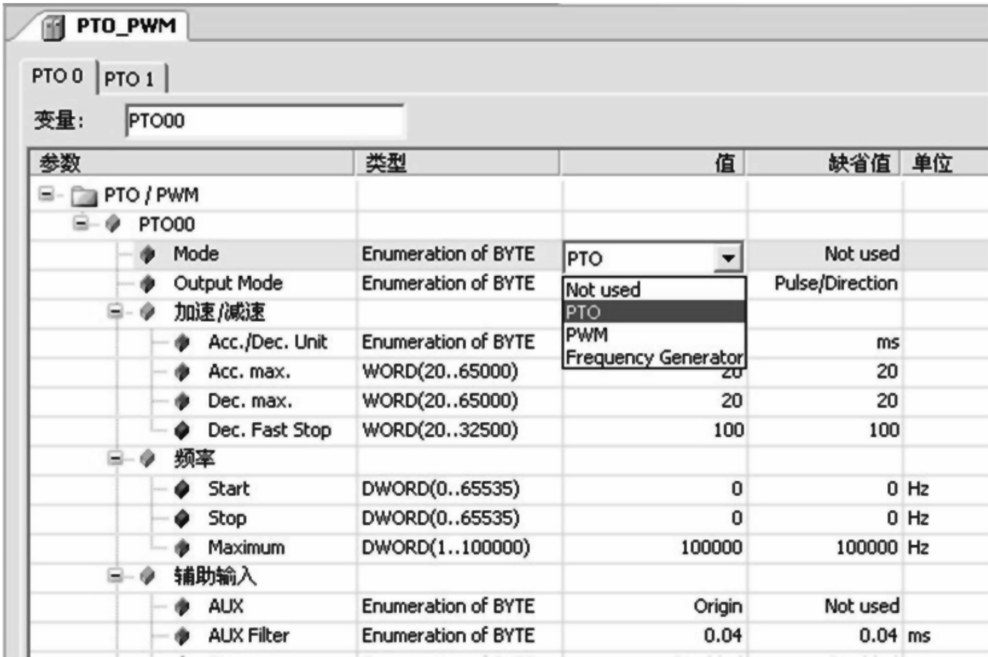


图 14-6　PTO/PWM 组态界面

出 PTO 用来做位置控制，或定义为脉冲宽度调制（PWM）来控制脉冲占空比可变的脉冲输出，或定义为频率发生器。当我们选定 PTO 后，接下来的栏目如加速/减速、频率、辅助输入就都会被激活。并且在变量栏被自动填入 PTO00。

我们也可以把其中一组 PTO 定义为脉宽调制 PWM，例如 PTO 1，如图 14-7 所示。

当我们在模式栏目中选定为 PWM 时，变量栏目被自动填为 PWM01。

点击此界面下方的 I/O 和摘要，会出现 I/O 功能汇总，如图 14-8 所示。

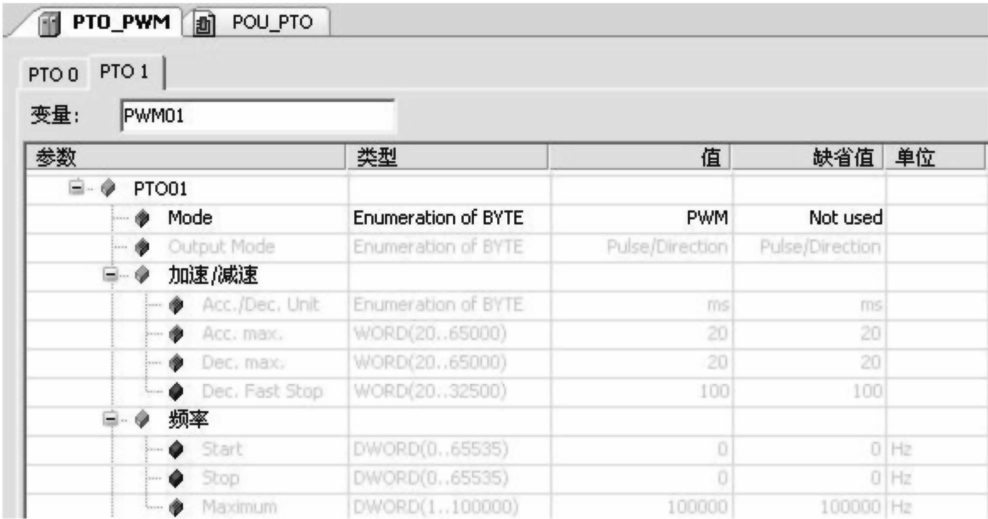


图 14-7　把脉冲输出定义为 PWM 模式

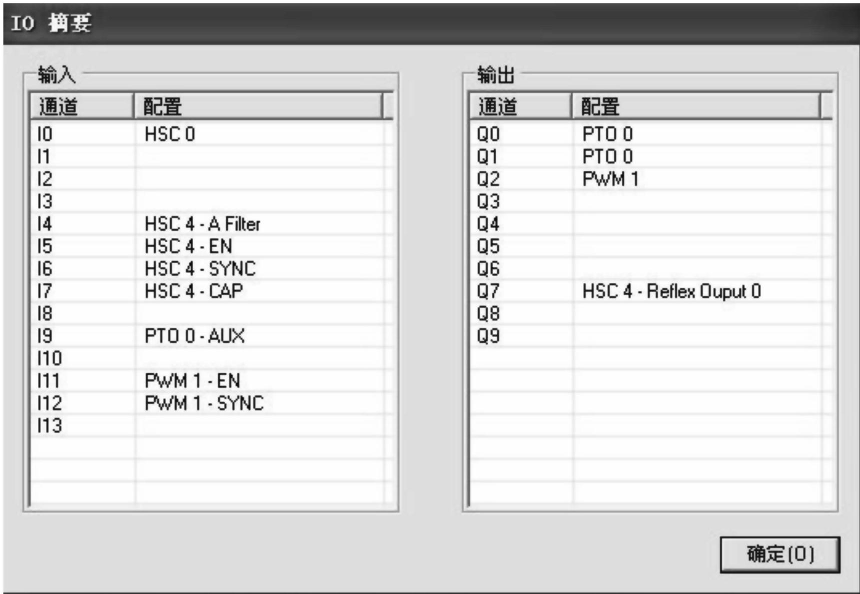


图 14-8　内置功能下的 I/O 功能汇总

在汇总图中，我们会看到 Q0、Q1 的输出为脉冲串输出。Q2 为脉宽调制输出点。输入点 I9 作为原点开关，而 I11、I12 则作为脉宽调制输出的使能信号和同步信号。

把 PTO 功能组态好后，我们就可以编辑程序，实现简单的运动控制功能了。首先要调入的功能块就是 PTOSimple，如图 14-9 所示。通过输入助手我们可以找到这个功能块并确定。这个功能块的作用是把组态好的硬件配置与驱动程序对应起来。所以这个功能块的命名不是随意的，而是调用已经组态好的变量名。

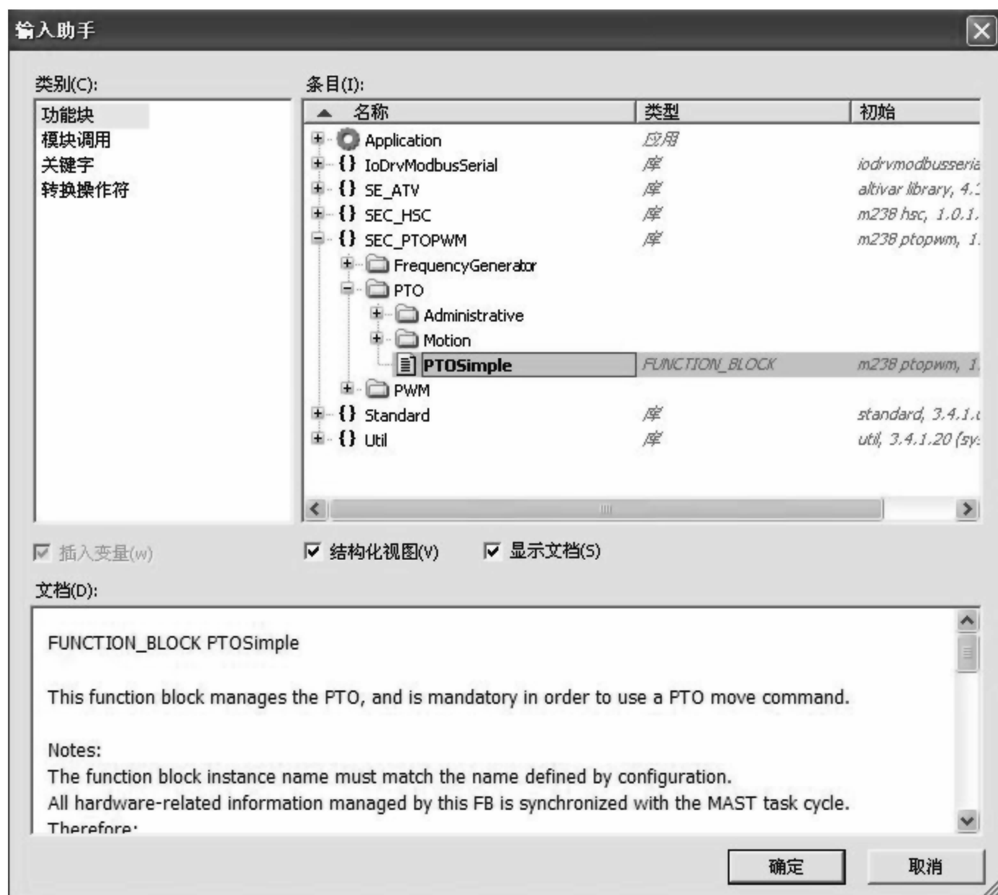


图 14-9　选择 PTO 功能块

如图 14-10 所示，在调入的功能块上方命名处，点击输入助手符号。

然后在弹出的输入助手框，找到定义好的变量 PTO00，如图 14-11 所示。

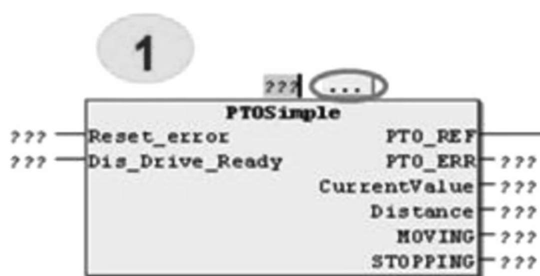


图 14-10　点击输入助手符号

确定后，功能块的名称自动形成，如图 14-12 所示。这也就意味着组态好的硬件结构与驱动轴对应完毕。当我们开始定义这个功能块的输出，例如 PTO_ REF 时，我们给它一个轴的名字 d1，那么，在接下来的运动程序中，只要 PTO_ REF 的名字是 d1，那么，组态好的硬件 PTO00 就要参与动作，Q0、Q1 输出的脉冲串就要驱动这个 d1 轴运动。

那么，d1 轴都可以做哪些运动呢？通常来说 d1 轴可以做 4 种基本运动。

1）恒速运动，即 d1 轴的速度控制，如图 14-13 所示。

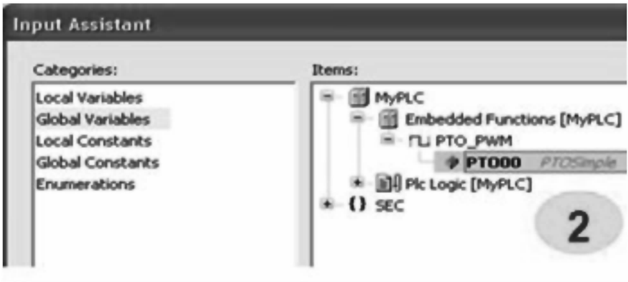


图 14-11　找到定义好的变量

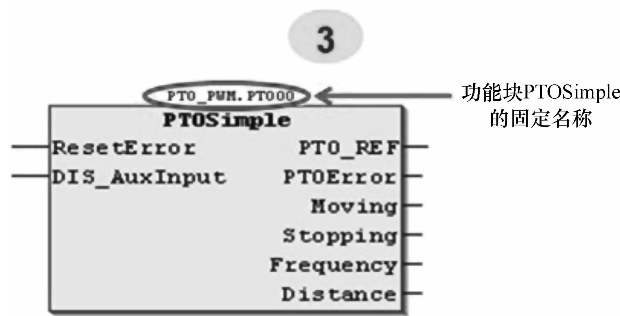


图 14-12　功能块名称形成

这种运动时，Q1、Q2 输出的脉冲先以一个启动频率把 d1 轴驱动起来，然后输出一个频率变化，使 d1 轴沿着一定的加速斜率不断升速，最终达到目标速度。在编程时，首先确定组态好的硬件与定义轴对应。即在功能块 PTOSimple 定义轴名称 d1，如图 14-14 所示。

然后在编程区调入速度运动功能块，借助输入助手找到相应功能块并确定，如图 14-15 所示。

确定调入速度功能块 PTOMoveVelocity，并在 PTO_ REF_ IN 处赋值 d1，表明是对 d1 轴的速度控制运动。k2 是一个启动开关，把目标速度赋值给 speed_ ref，当 k2 闭合，则 d1 以 100Hz/ms 升速，达到目标速度后进行恒速运动，并且 InVelocity 输出。恒速运动功能块如图 14-16 所示。

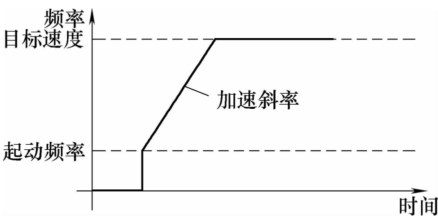


图 14-13　恒速运动

2）停止运动，如图 14-17 所示。

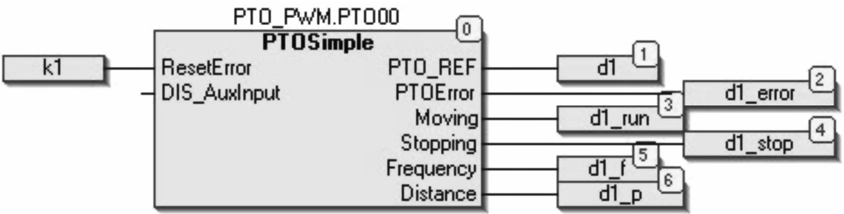


图 14-14　定义轴名称



图 14-15　调入相应功能块

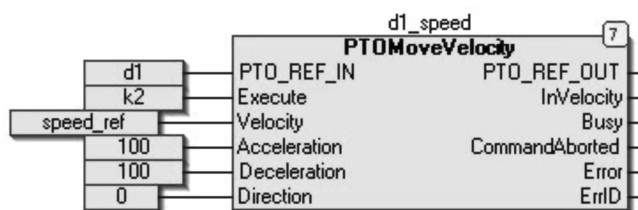


图 14-16　恒速运动功能块

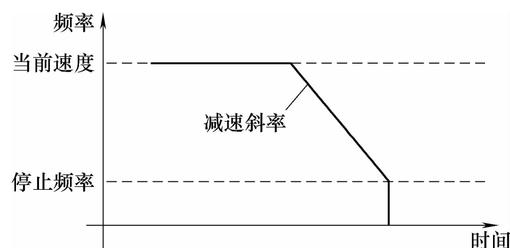


图 14-17　停止运动

在 d1 轴的运动过程中接到停止命令，先以一个减速斜率降速，达到停止频率后，d1 轴运动停止。在编程区调入停止功能块 PTOStop 来完成这个功能，如图 14-18 所示。开关 k3 触发一个停止命令。停止后 Done 信号输出。

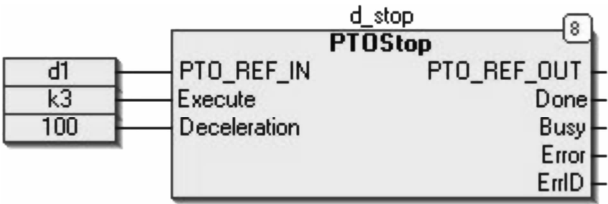


图 14-18　停止功能块

3）相对位置运动，即点到点的运动，如图 14-19 所示。

这种运动是点到点的相对位置运动。Q1、Q2 输出的脉冲个数一定，并且启动时沿一定的加速斜率升速，快到位时，按一定的减速斜率降速，最终达到设定位置。在编程时，调入 PTOMoveRelative 功能块来完成这个功能。点到点的距离赋值给 pos_ ref 变量。k4 启动这个 d1 轴的位置运动。到位后 Done 信号输出。位置运动功能块如图 14-20 所示。

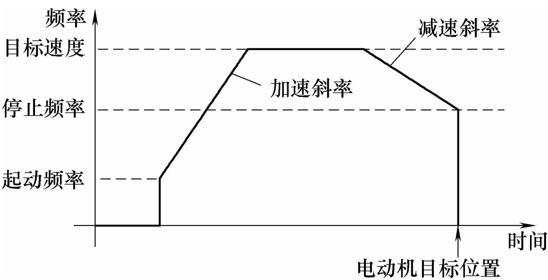


图 14-19　相对位置运动

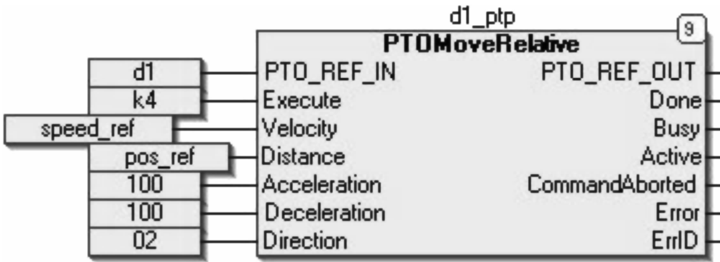


图 14-20　位置运动功能块

4）寻原点运动，即寻原点，定基准位置，如图 14-21 所示。

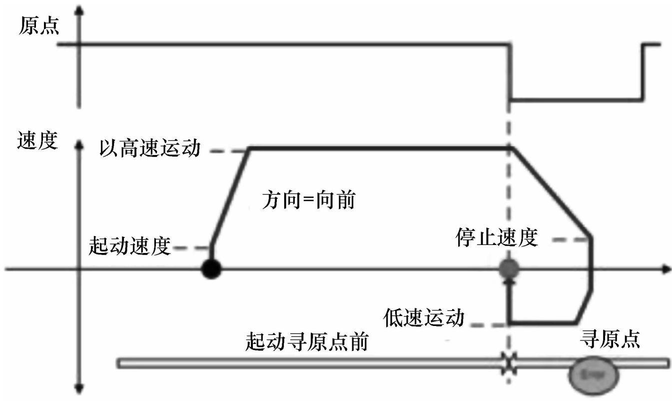


图 14-21　寻原点运动

寻原点运动需要一个外部开关 Origin，这需要在组态时就定义好，如 PTO/PWM 组态画面那样，在辅助输入栏选择 Origin，然后在内置功能下的 I/O 功能汇总中，就能看到定义的辅助输入点 I9，注意这个点要接为常闭接点，不然的话程序会报警。如图 14-21 所示，启动寻原点运动后，d1 轴的运动先以一个高速向原点运动，撞开原点开关 I9 后降速并反向以一个低速离开原点运动，当原点再次闭合后运动停止，并以此点作为原点。位置值清零。在组态时，寻原点的方向可以定义为向前 Cam forward 或向后 Cam backward。

在编程时调入相应功能块 PTOHome，如图 14-22 所示。k5 启动一个寻原点命令。

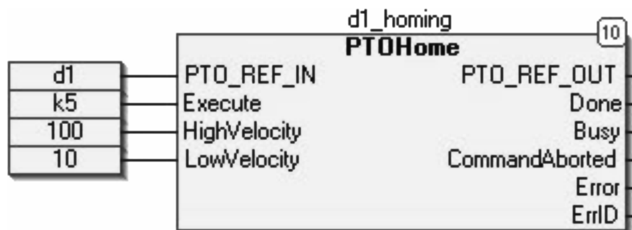


图 14-22　寻原点运动功能块

在通过 PTO 运动的过程中，我们可以在线观察或修改运动参数，也可以通过在线诊断功能块找出错误所在。应用这些功能块时同样可以借助输入助手，把相应功能块调入。例如：调入读取参数功能块 PTOGetParm，PTO 管理集群如图 14-23 所示。



图 14-23　PTO 管理集群

确定后，读取功能块如图 14-24 所示。

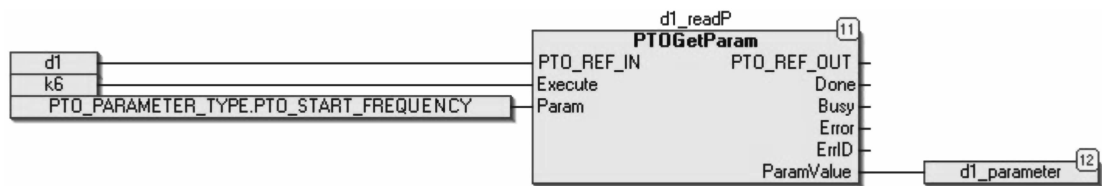


图 14-24　读取功能块

在读取功能块中 Param 是一个枚举变量 PTO_PARAMETER_TYPE。

这个枚举变量包含如下参数，见表 14-1。

表 14-1　枚举变量 PTO_PARAMETER_TYPE

枚举器	值	说　明
PTO_START_FREQUENCY	00(十六进制)	PTO 运动的启动速度
PTO_STOP_FREQUENCY	01(十六进制)	PTO 运动的停止速度
PTO_EMY_DEC	02(十六进制)	PTO 紧急停止的减速度

因此变量后边跟一个“. ”则会下拉所有枚举的参数。在上例中，我们要读出启动频率。所以给 Param 赋值 PTO_PARAMETER_TYPE. PTO_START_FREQUENCY，k6 闭合，启动频率被读出并输出给 d1_parameter。当然，我们也可以直接给 Param 赋值 00 读取启动频率，或赋值 01 读取停止频率，或赋值 02 读取紧急停止斜率。

同理，我们也可以在线修改启动频率、停止频率和紧急停止斜率。用到的功能块如图 14-25 所示。这个功能块把启动频率修改为 20。

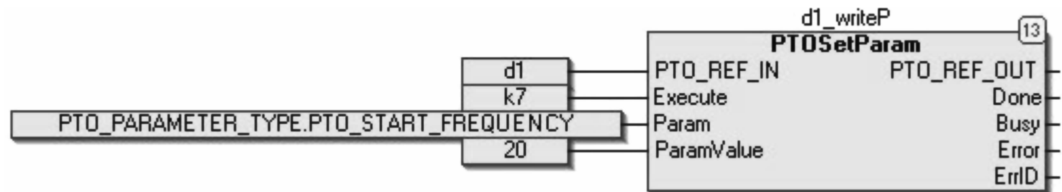


图 14-25　参数写操作功能块

诊断错误功能块如图 14-26 所示。当 k8 闭合，诊断信息输出给变量 d1_PTODiag。

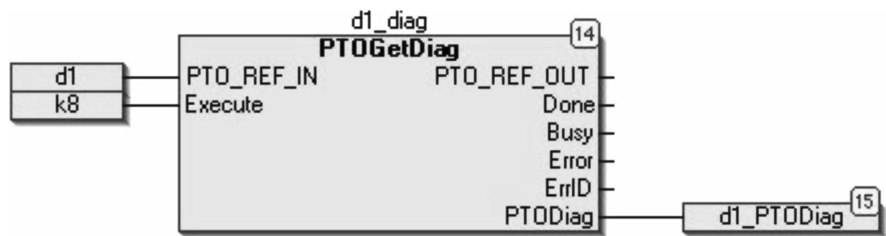


图 14-26　诊断错误功能块

根据变量 d1_ PTODiag 输出位的变化，可以知道错误情况，见表 14-2。

表 14-2　诊断输出信息对照

DWORD 位	含　义	DWORD 位	含　义
0...3	未使用	21	检测到回归错误
4	检测到内部错误	22	频率无效
5，6	未使用	23	加速度无效
7	检测到配置错误	24	减速度无效
8...16	未使用	25	命令被拒绝
17	驱动器未就绪（辅助输入 DriveReady 为 FALSE）	26	方向无效
18～20	未使用	27～31	未使用

综上所述，我们知道了采用 PTO 方式进行运动控制的一组基本功能块。需要注意的是，在采用 PTO 功能时，首先要进行方式组态，然后要把功能块上的设备和组态硬件联系起来。这样才能使电动机轴运转起来。同理，我们也可以把另一个通道组态成 PTO，这样我们就可以驱动两个电动机的运动。好了，做一下练习找找感觉吧。

14.2　采用 CANopen 总线控制的设计方法

我们在第 12 章讨论过采用 CANopen 总线控制多个设备包括伺服驱动。在这一节我们将讨论基于 CANopen 总线的运动控制功能块如何使用。

如图 14-27 所示，我们

- 首先添加和组态 CANopen 总线界面；(1)
- 在总线界面下添加驱动设备，如 LEXIUM05 和 LEXIUM32(2)。

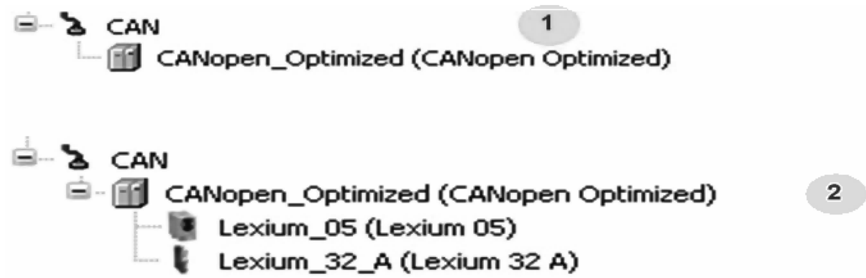


图 14-27　添加总线界面和添加驱动设备

在基于 CANopen 总线下，我们有以下单轴运动功能块，如图 14-28 所示。

在程序区，调入一个运算块，如图 14-29 所示。点击问号旁的输入助手符号。

打开输入助手，如图 14-30 所示，选择一个单轴运动功能块，如 MC_Jog_LXM，按“确定”键。

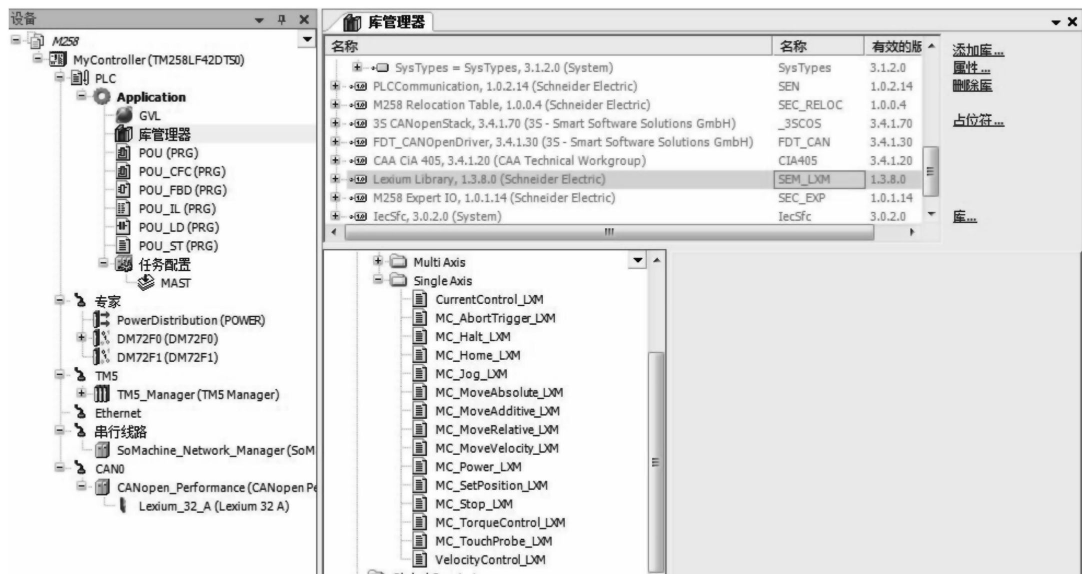


图 14-28　基于 CANopen 总线的单轴运动功能块



图 14-29　调入运算块

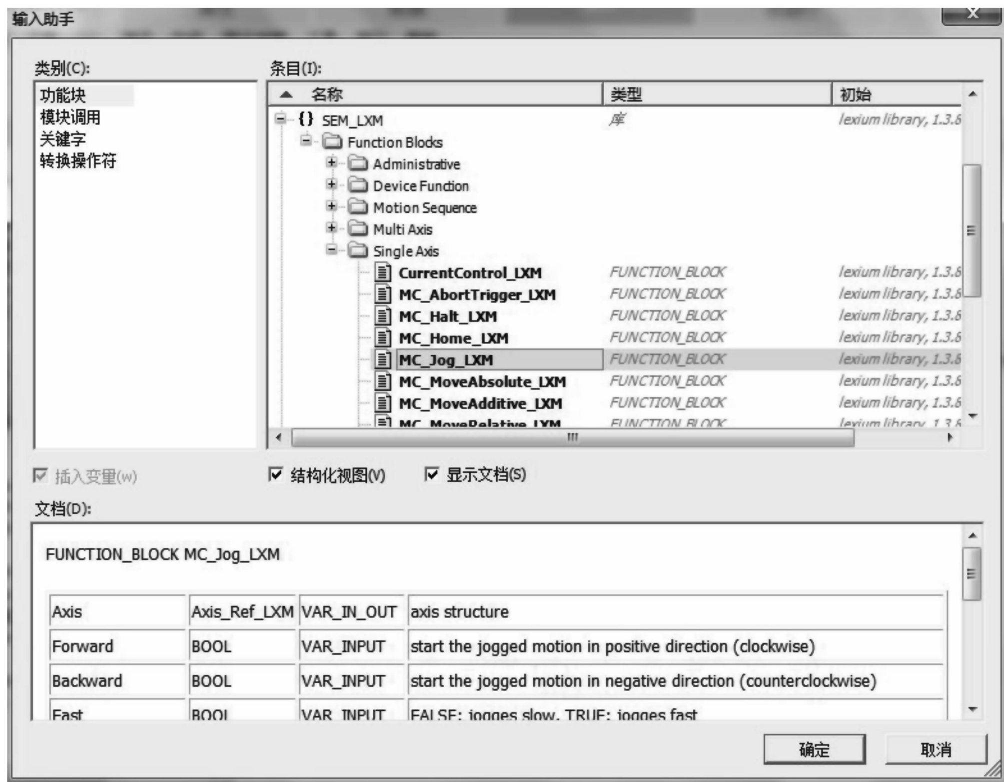


图 14-30　在库中选择功能块

调入此功能块，如图 14-31 所示，这是一个电动机轴的点动功能块。

我们可以配置好功能块的输入变量，如图 14-32 所示。在功能块的输入点 Axis 配置运动轴的名字。例如，在图 14-27 中添加的驱动设备 Lexium32A 的轴名称为 Lexium_32_A，所以，配置轴的输入变量就是 Lexium_32_A，如图 14-32 所示。其他输入按照说明配置即可。d1_forward 为电动机向前按钮，d1_backward 为向后按钮，d1_fast 为快/慢速切换按钮。

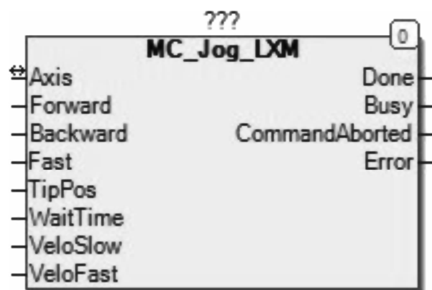


图 14-31　电动机轴的点动功能块

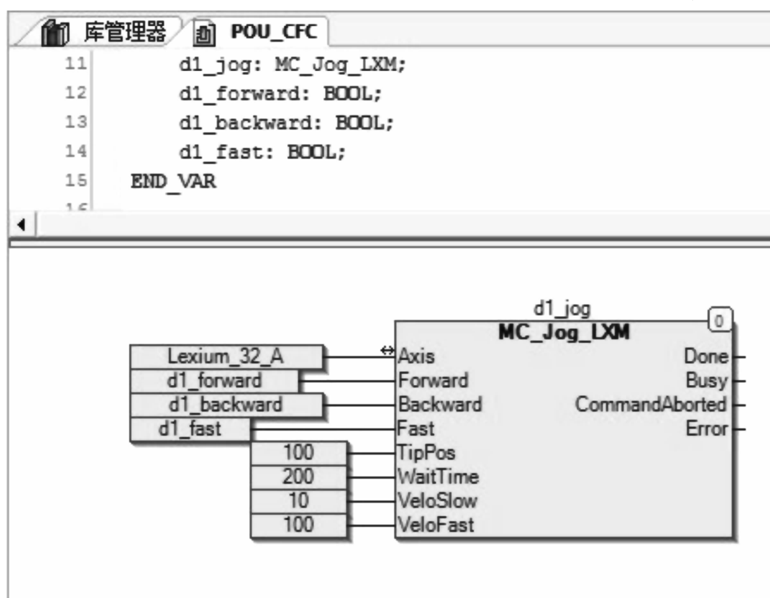


图 14-32　配置好功能块的输入变量

在操作这些动作时，我们首先要给电动机使能，这就要用到 MC_Power_LXM。同理，调入此功能块并配置好输入，如图 14-33 所示。当开关 K 闭合时，配置在总线上的电动机 Lexium_32_A 使能。

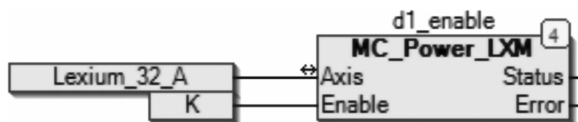


图 14-33　电动机轴使能功能块

需要电动机寻原点时，调入寻原点功能块，如图 14-34 所示。在 HomingMode 输入点选择寻原点方式，然后填好其他参数，就可以运行这个功能块了。

做点到点运动时，我们可以执行相对运动和绝对运动两种模式。位置模式功能块如图 14-35 所示。

做叠加运动时，我们可以调入 MC_MoveAdditive_LXM 来实现。例如，我们在做一个点到点运动时，需要在这段距离中有不同的速度。如图 14-36 所示程序，在位置运动功能块，我们给走的距离和速度赋值，如果要在走一定距离时，以不同的速度进行，则在 MC_MoveAdditive_LXM 功能块的距离 Distance 上赋值“0”。速度赋值写上改变的数值。

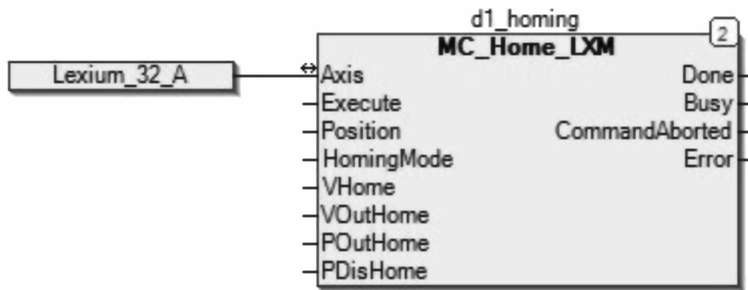


图 14-34　寻原点功能块

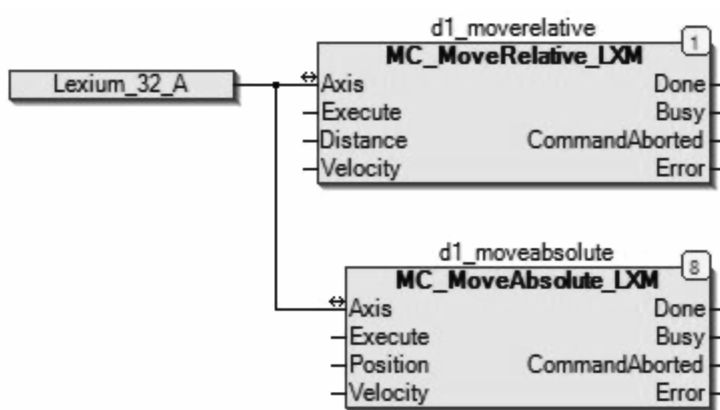


图 14-35　位置模式功能块

注意：改变数值的生效是在 k6 的上升沿触发。如图 14-37 所示，图的上边曲线为速度变化曲线，下边曲线为位置变化曲线。速度变化的触发是由 k6 的上升沿引起。

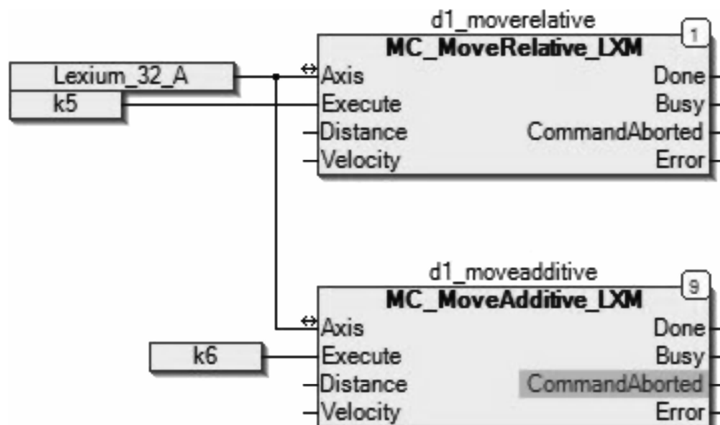


图 14-36　叠加运动程序

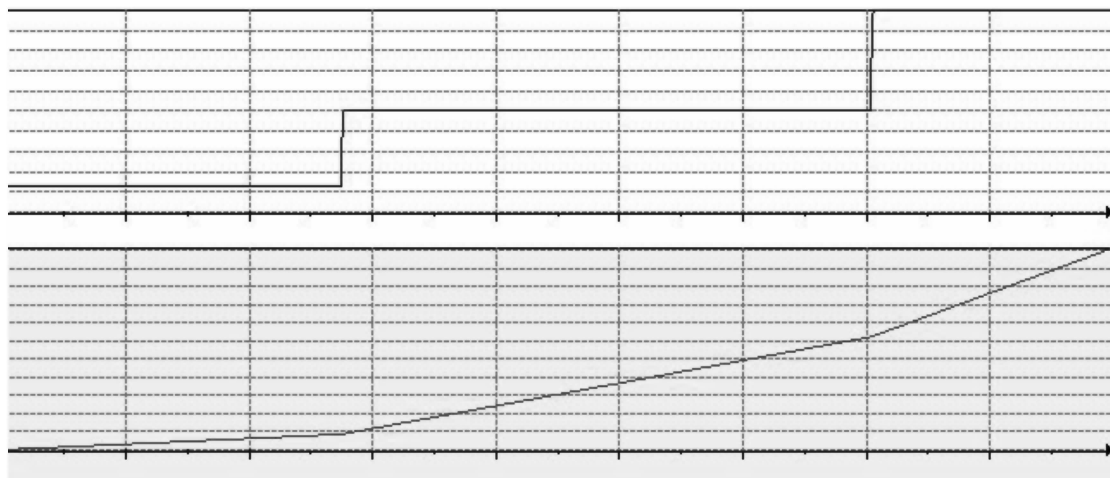


图 14-37　速度叠加运动

如果要在走完一个距离后，再以不同的速度叠加一个距离，则在 MC_ MoveAdditive_ LXM 功能块的距离 Distance 上赋值“叠加的距离”。速度赋值写上走这段距离的速度值。速度值的改变是由 k6 的上升沿触发，走的位置是位置功能块的距离值加上叠加功能块的距离值。位置叠加运动如图 14-38 所示。

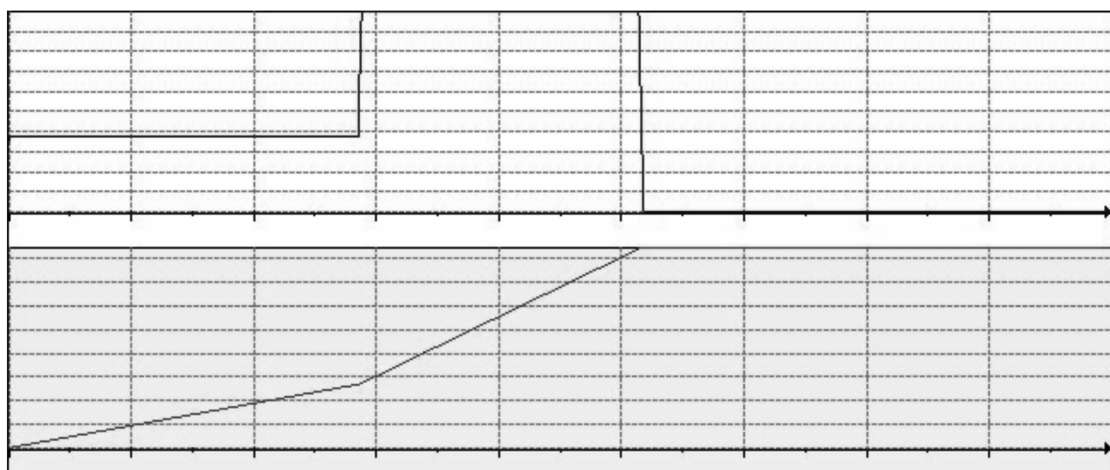


图 14-38　位置叠加运动

当驱动器出现问题或对驱动器的状态进行监控时，我们可以调用如图 14-39 所示的驱动器监控功能集群。

例如，对驱动器进行复位操作，调用 MC_Reset_LXM。

按“确定”键，调入复位功能块，如图 14-40 所示。k7 上升沿触发复位。

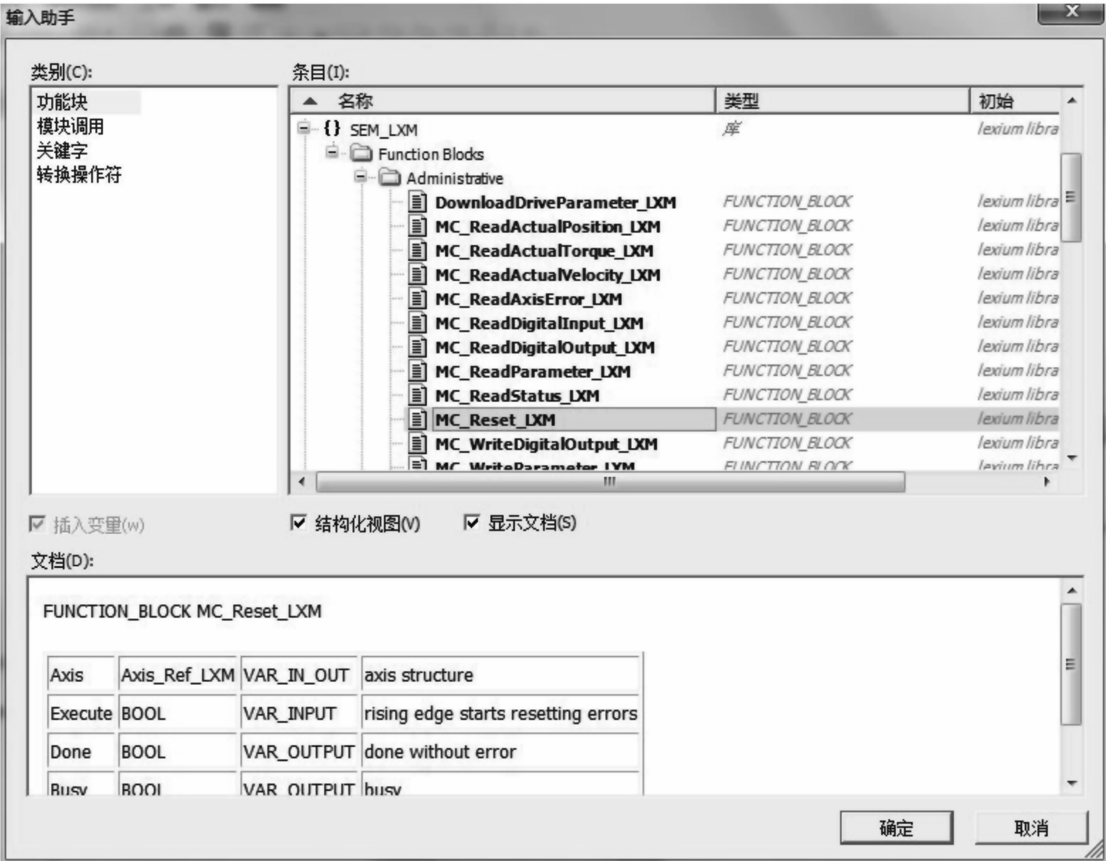


图 14-39　驱动器监控功能集群

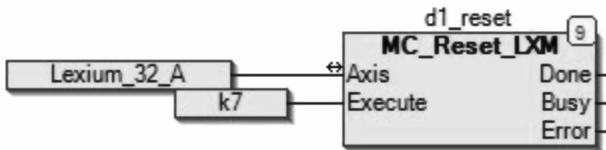


图 14-40　复位功能块

当我们需要操控驱动器内部的运动任务时，我们可以调用如下内部任务集群的功能块来完成对驱动器内部任务的操控，如图 14-41 所示。

需要注意的是：可以有内部运动任务的驱动器是 Lexium32M，所以在组态硬件时，要添加的设备是 Lexium32M，如图 14-42 所示。

在启动运动任务功能块 StartMotionSequence_ LXM 时，只要把内部运动任务号赋值给 DataSetNumber，然后执行即可驱动一个内部运动任务。而读写运动任务却是需要一组参数的交换。因此读/写功能块中的 DataSet 是一个结构变量。例如，如图 14-43 所示，我们命名 s1、s2 为这个结构变量。

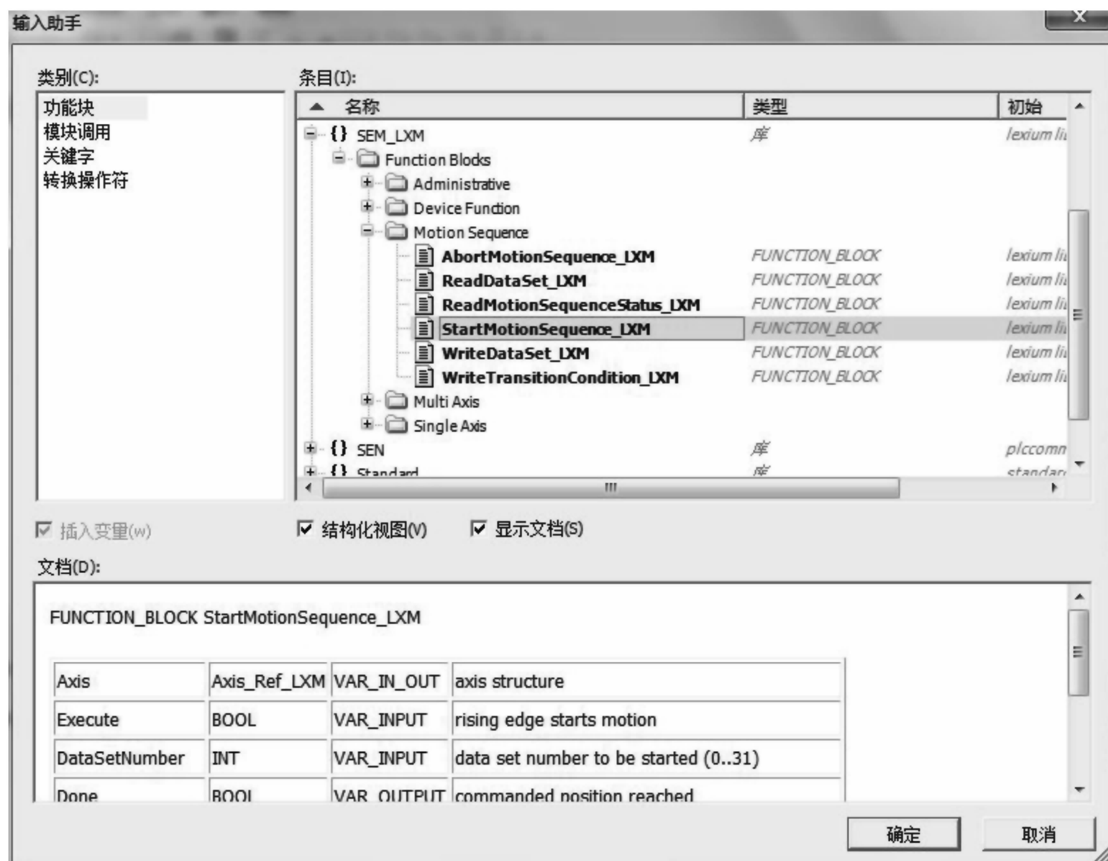


图 14-41　驱动器内部任务操控集群

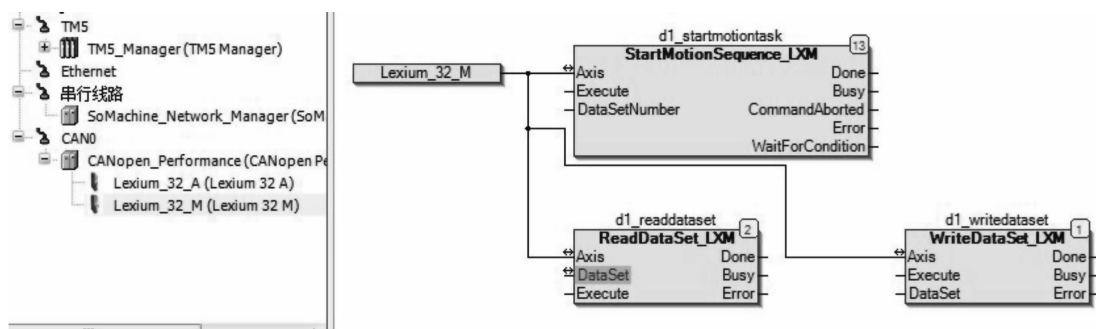


图 14-42　通过总线操控驱动器内部任务

当 s1 后面加“.”时，它下拉的参数就是这个结构体下的所有变量，如图 14-44 所示。

通过对这些变量的写操作，可以随心所欲地修改内部运动任务的运动轨迹和执行条件，

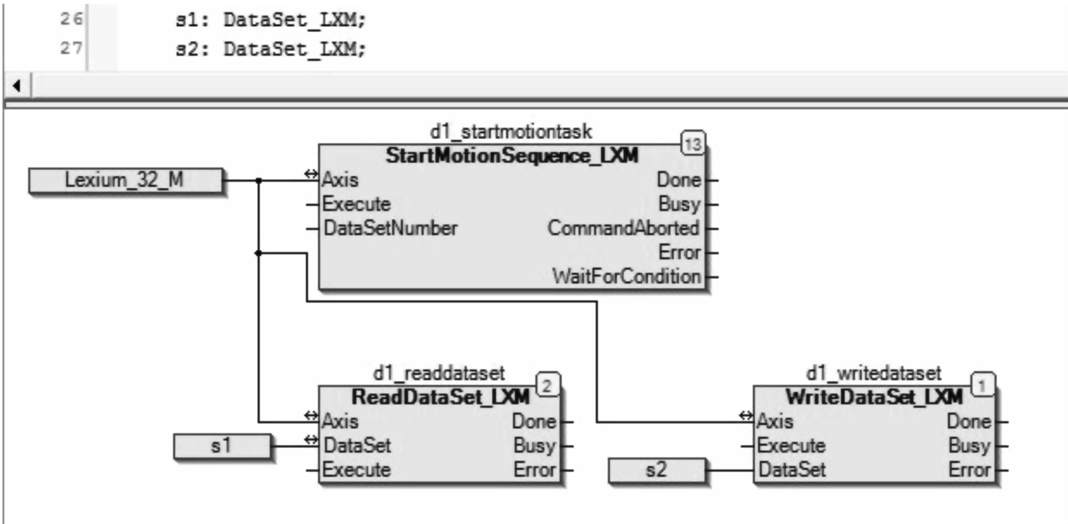


图 14-43　命名结构变量

无限地扩展了运动任务的规模。对这些变量的写操作程序如图 14-45 所示。

这些运动任务在驱动器调试软件下的编辑如图 14-46 所示。

通过这个事例，我们知道驱动器内部任务的编辑不仅可以用驱动器调试软件来编辑，也可以在 SoMachine 程序中编辑使用，也就是说我们把运动任务放在驱动器里，而采用程序来启动执行运动序列，它的好处是不占用控制器的 CPU 资源，使运动更加迅速。

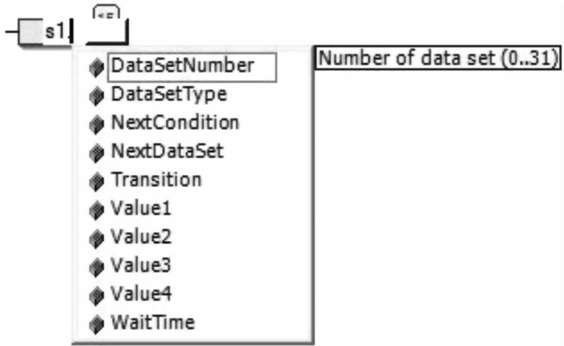


图 14-44　组成内部运动任务的所有参数

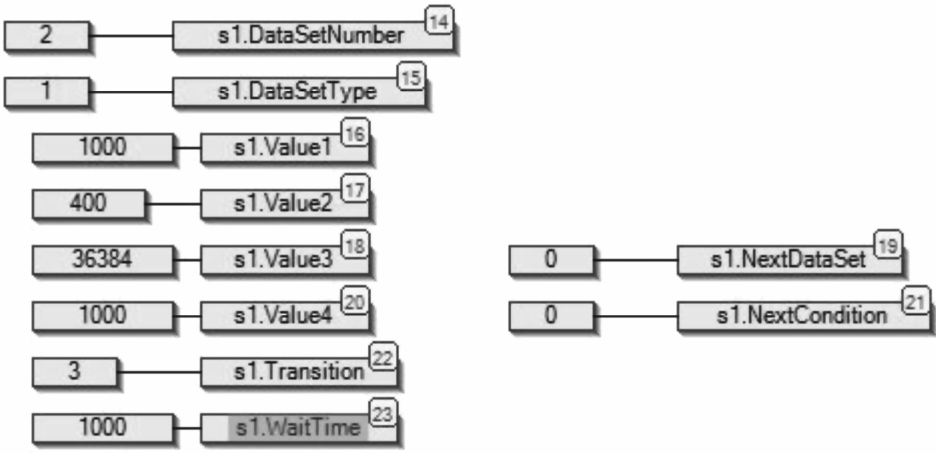


图 14-45　对变量的写操作程序

File Edit Connection Parameter Functions Diagnostics Safety Functions Window ?

Modbus

No	Type	Setting A	Setting B	Setting C	Setting D	Transition	Subsequent data set	Transition condition 1	Transition value 1
0	Reference Movement	100	0			Abort And Go Next	1	Continue Without Condition	0
1	Move Absolute	1000	200	16384	1000	Blending Previous	2	Continue Without Condition	0
2	Move Absolute	1000	400	36384	1000	Buffer And Start Next	0	Wait Time	1000
3	None					No Transition	0	Continue Without Condition	0
4	None					No Transition	0	Continue Without Condition	0

图 14-46 内部运动任务的编辑

14.3 采用 CANmotion 总线控制的设计方法

在 CANopen 总线上连接的伺服驱动只能做单轴的独立运动，不能做有关联的同步轴运动，例如做电子齿轮和电子凸轮的应用以及 CNC 的差补运动。因此，为了满足对同步运动的需求，在运动控制器上我们开发了专门做同步运动的 CAN 总线，称之为 CANmotion 总线，它的物理结构与 CANopen 总线一样，但具有设备的同步功能。这样就可以做多轴同步了。如图 14-47 所示，我们

- 首先添加和组态 CANmotion 总线界面；
- 在总线界面下添加驱动设备，如 d3 和 d4；
- 当然，在运动控制器中我们还能添加虚拟轴 d1、d2。

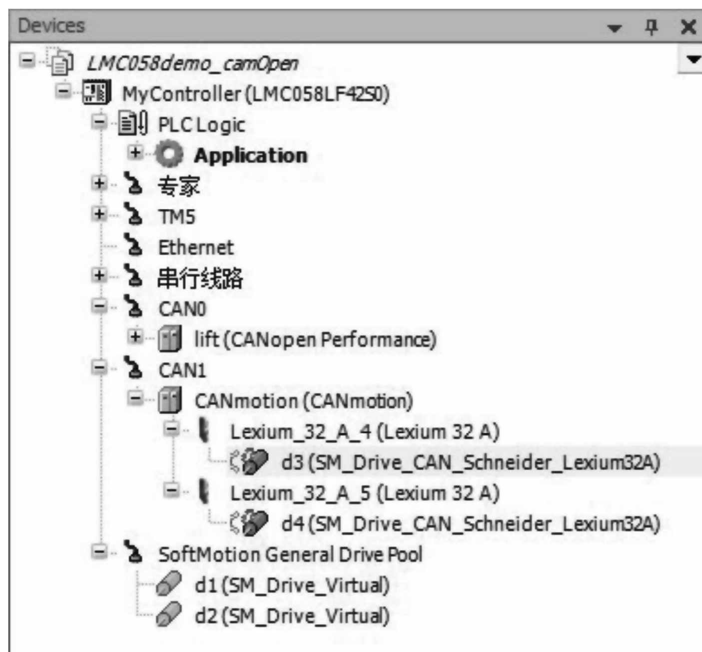


图 14-47 CANmotion 总线配置

在 CANmotion 总线，使用的运动控制库集群如图 14-48 所示。

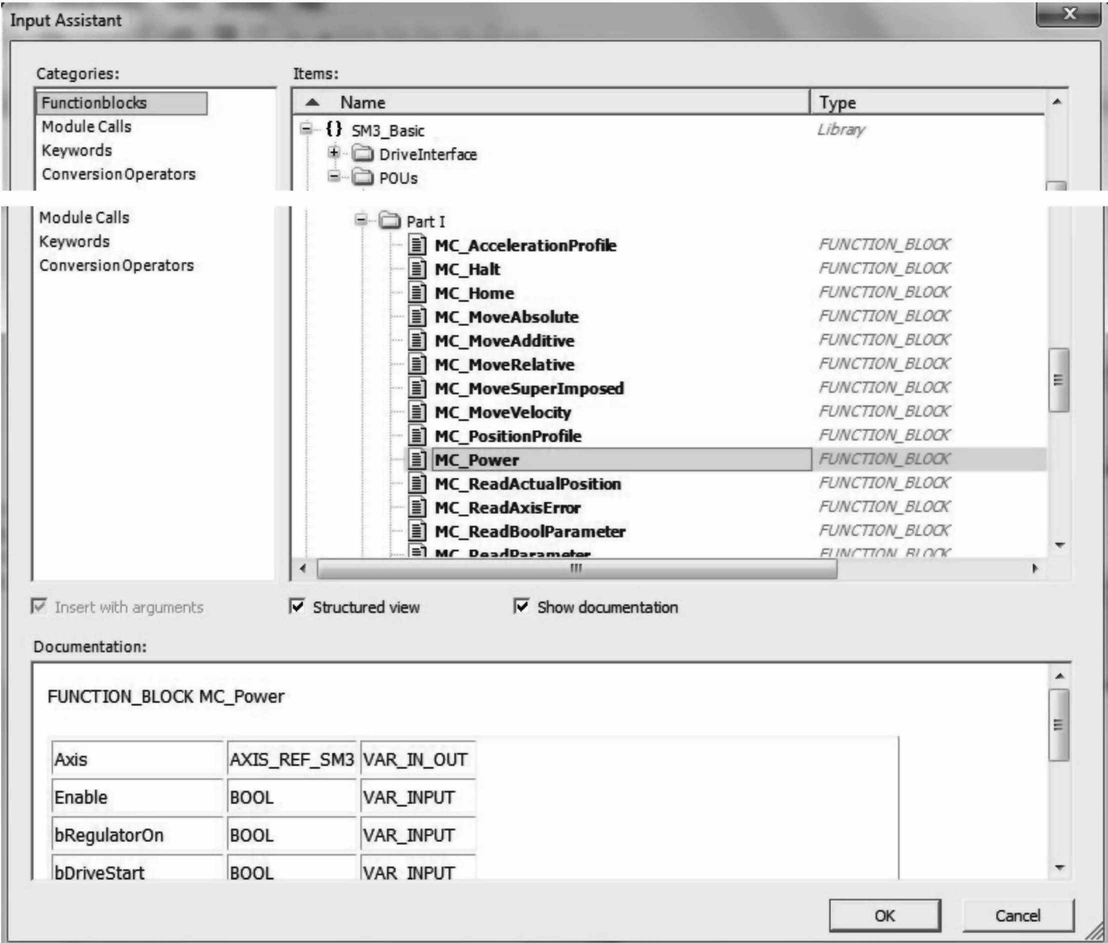


图 14-48　在 CANmotion 总线上的伺服驱动功能库集群

例如，我们使能一个伺服轴，调入的功能块如图 14-49 所示。

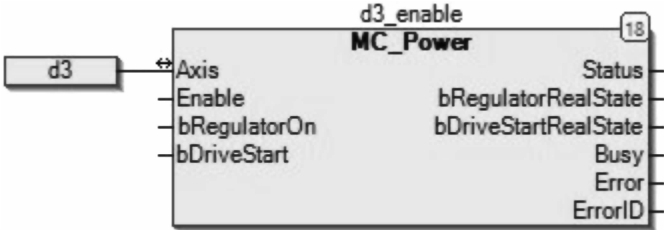


图 14-49　使能一个伺服轴的功能块

我们看到在 CANmotion 总线下，使能一个伺服轴需要 3 个开关量，即需要顺序执行 Enable、bRegulatorOn、bDriveStart。当我们需要两个轴或多个轴同步时，可以调用多轴同步功能集群下的功能块，如图 14-50 所示。

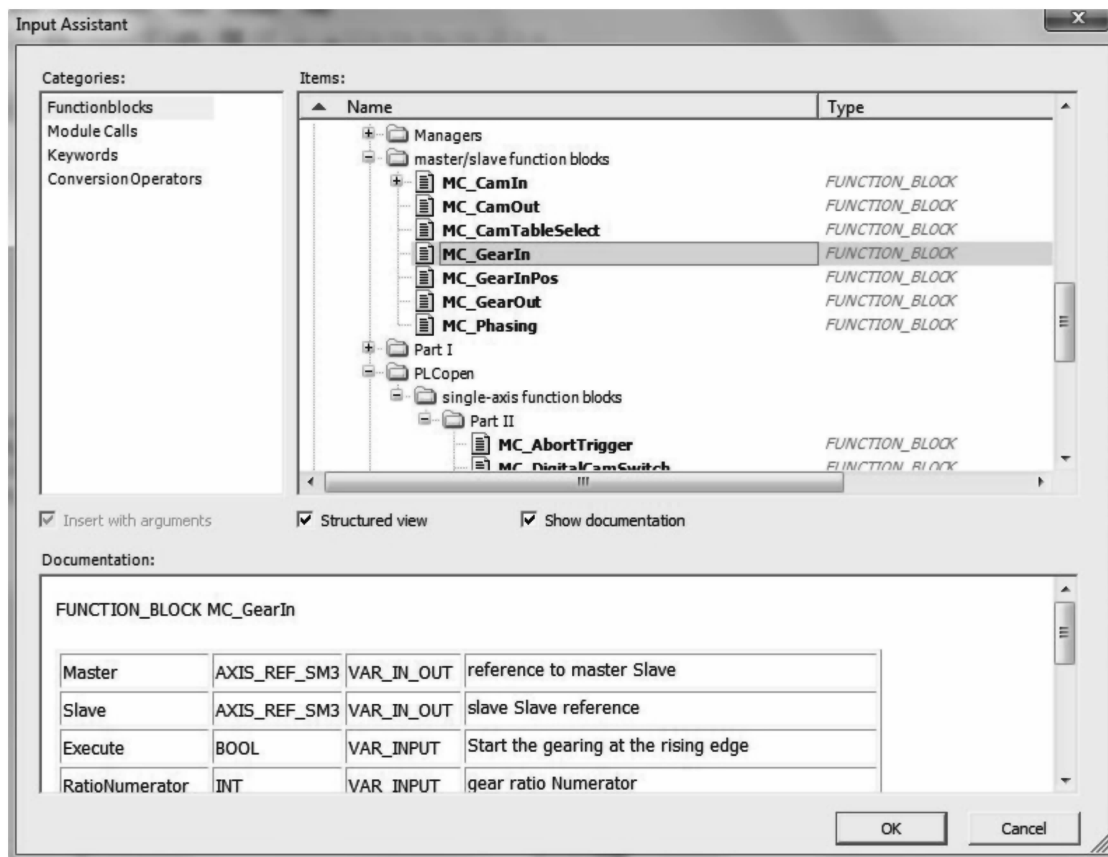


图 14-50　同步功能块集群

例如，我们要使轴 d3 作为主轴，轴 d4 跟随轴 d3 运动。我们调入电子齿轮功能块如图 14-51 所示。

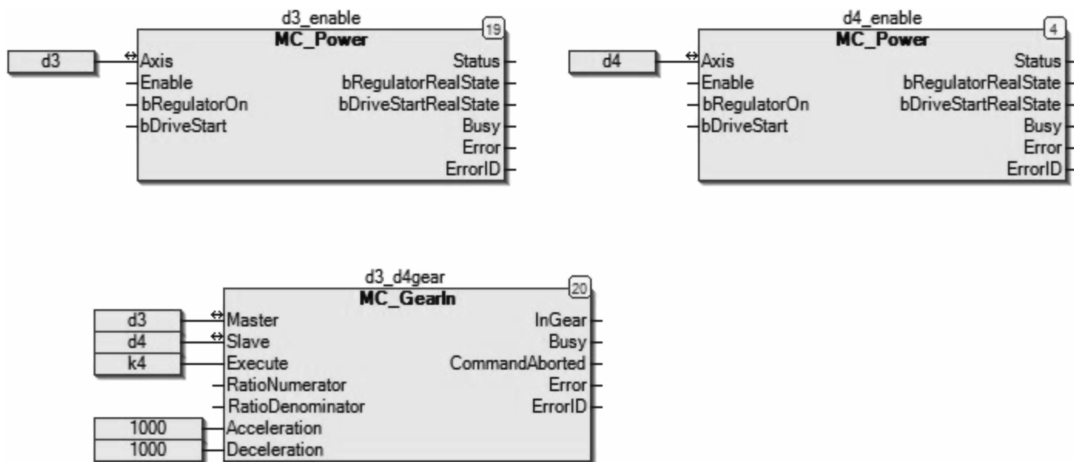


图 14-51　电子齿轮功能块

把轴 d3、d4 使能后，只要把 k4 闭合，inGear 输出为“真”则这两个轴就耦合了。这时轴 d3 运动，轴 d4 就跟着运动，跟随的距离比例由电子齿轮比决定，即参数 RatioNumerator 分子，参数 RatioDenominator 分母。这个比例是可以在线修改的。需要注意的是：通常在运动时，分母是不允许修改的，只能修改分子。

要使两个轴脱离耦合，我们可以使用如图 14-52 所示功能块。

在操作脱离耦合时，一种状态是静止脱离。即在主轴 d3 停止状态下，k5 的上升沿触发离合，使主从轴分离。这时从轴 d4 也是停止的。脱离后轴 d3 可以单独执行各种运动。还有一种情况是轴 d3 在运动状态，从轴 d4 跟随运动，这时 k5 的上升沿触发离合，使轴 d4 脱离轴 d3 ，这时的从轴 d4 是运动的，它的速度就是离合打开时的速度。耦合脱开后，轴 d3 速度再变化将不影响轴 d4 的速度变化。如果要停止轴 d4 的运动，就需要调入停止功能块来操作。

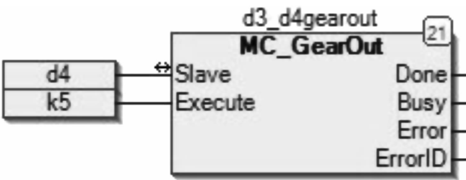


图 14-52　脱离耦合功能块

在同步功能集群中，还有一个功能是控制 2 个轴或多个轴的相位同步，即使几个轴的运动总是保持一个固定的位置差，这个功能如图 14-53 所示。

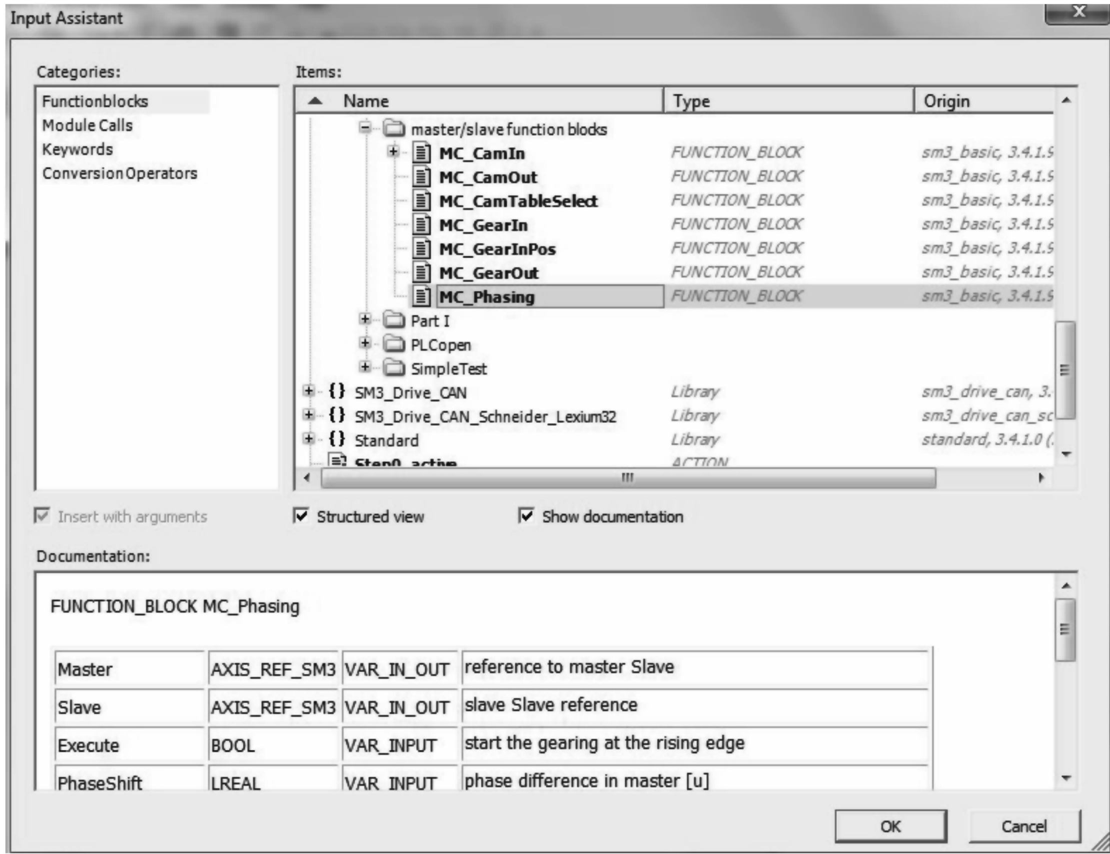


图 14-53　相位同步功能

当 k6 闭合后，2 轴耦合并使同步运动的相位差保持在 90°，如图 14-54 所示。

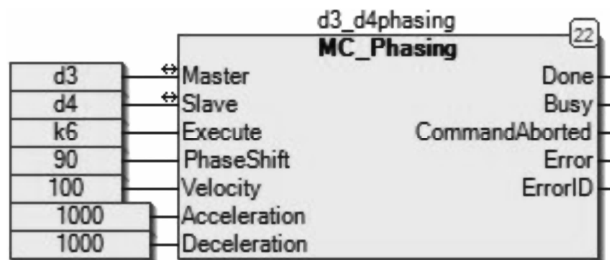


图 14-54　相位差 90°的同步

当修改相位差值后，需要 k6 再次闭合，使两个轴的同步达到新的相位差值。相位差值的修改是可以在线运行的。

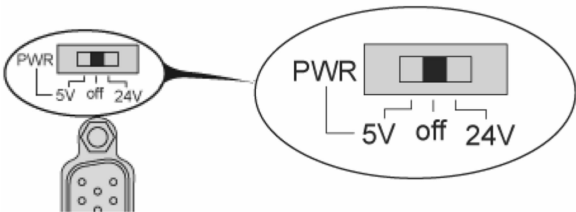
当然，在同步运动类型中，还有 CAM 电子凸轮的运动和 CNC 差补运动，这些运动的应用我们将在《施耐德 SoMachine 控制器的应用及编程指南》高级篇中详细讲解。

第 15 章　运动控制器中编码器的应用

15.1　编码器的硬件接线

运动控制器中编码器的应用是大量存在的，以运动控制器 LMC058 为例，其本体就有一个编码器接入口，编码器的供电可以是 5V 和 24V。编码器供电选择和接线如图 15-1 所示。

下图显示了开关的三个不同位置：



用户可通过开关选择下列任一电压：

- +5 Vdc
- 不提供配电
- +24 Vdc

下表显示了提供给引脚 7 和 15 的电压：

编码器接口开关位置	引脚 15	引脚 7
5V	5 Vdc	0 Vdc
关	0 Vdc	0 Vdc
24V	0 Vdc	24 Vdc

图 15-1　编码器供电选择和接线

控制器可以接入的编码器有两种类型：一种是增量型差分信号输入（incremental），最大输入频率 1M；还有一种类型是串行绝对编码器（SSI），输入的最大频率是 200kHz。编码器类型图示如图 15-2 所示。

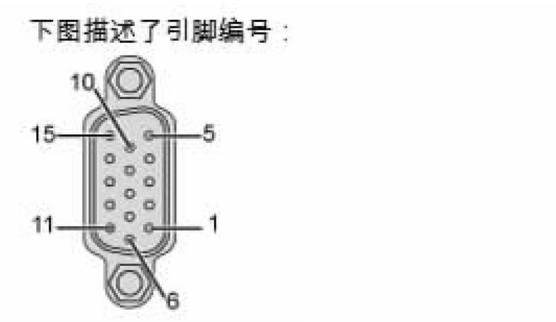
控制器本体编码器口引脚分配如图 15-3 所示。

增量编码器的接入如图 15-4 所示。

绝对编码器的接入如图 15-5 所示。

特性	描述	
递增	信号类型	A+、A-、B+、B-、Z+、Z-
	最大工作频率	200 kHz/每输入 x 4，或 1 MHz（用于计数）
SSI	时钟电压	5 Vdc
	最大时钟频率	200 kHz

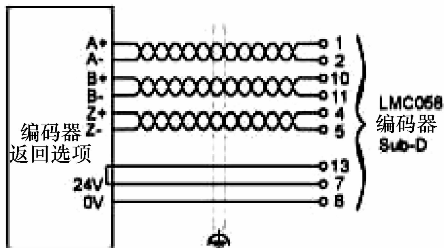
图 15-2　编码器类型图示



描述	编码器	引脚	VW3M4701 可选线颜色
递增编码器	A+	1	红色/白色
	A-	2	棕色
	Z+	4	橙色
	Z-	5	黄色
	B+	10	白色
	B-	11	紫色
绝对 SSI 编码器	SSI 数据 +	1	红色/白色
	SSI 数据 -	2	棕色
	CLKSSI +	6	绿色
	CLKSSI -	14	棕色指示灯
5 V 编码器电源	+ 5 Vdc	15	紫色指示灯
	0 Vdc	8	粉色
24 V 编码器电源	+ 24 Vdc	7	蓝色
	0 Vdc	8	粉色
编码器配电反馈	电源回路	13	绿色指示灯
屏蔽层		机壳	电缆编织屏蔽

图 15-3　编码器口引脚分配

下图描述了安装在编码器接口上的编码器 (RS - 422 / 24 Vdc) 的接线图：



下图描述了安装在编码器接口上的编码器 (RS - 422 / 5 Vdc 或推挽式) 的接线图

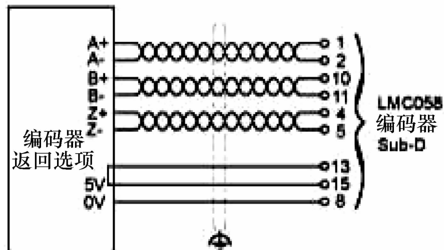


图 15-4　增量编码器的接入

下图描述了安装在编码器接口上的编码器 (SSI) 的接线图

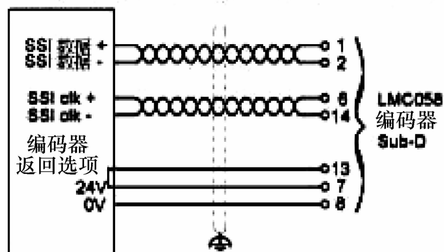


图 15-5　绝对型编码器的接入

15.2　LMC058 中编码器的应用

打开专家栏目，选择 Encoder 点击鼠标右键，添加编码器。

选择 Motion Encoder：命名 Encoder 名字为 SoftMotion_Enc。编码器添加及组态如图 15-6 所示。

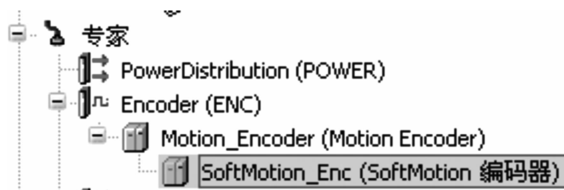


图 15-6　编码器添加及组态

双击“Motion_Encoder”组态编码器功能。可以选择位置捕捉 CAP，motion_encoder 支持 2 个快速输入点捕捉位置，完成 CAP0、CAP1 两个单点捕捉位置及双点捕捉距离。组态编码器功能如图 15-7 所示。

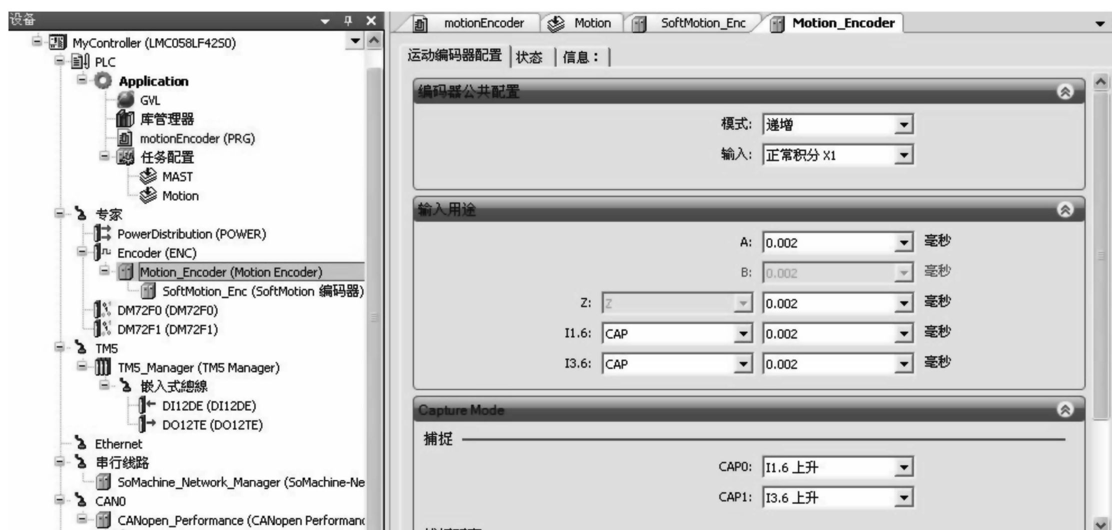


图 15-7　组态编码器功能

双击 SoftMotion_Enc (SoftMotion 编码器)组态编码器量程和运动类型，如图 15-8 所示。



图 15-8　组态编码器量程和运动类型

1. Motion encoder 运动编码器应用编程

与主轴编码器组成电子齿轮关系

Motion_encoder 可以使用 SM3_Basic 的应用库。

主轴编码器的名字为 SoftMotion_Enc，编码器从主轴编码口进入。与编码器组成电子齿轮如图 15-9 所示。

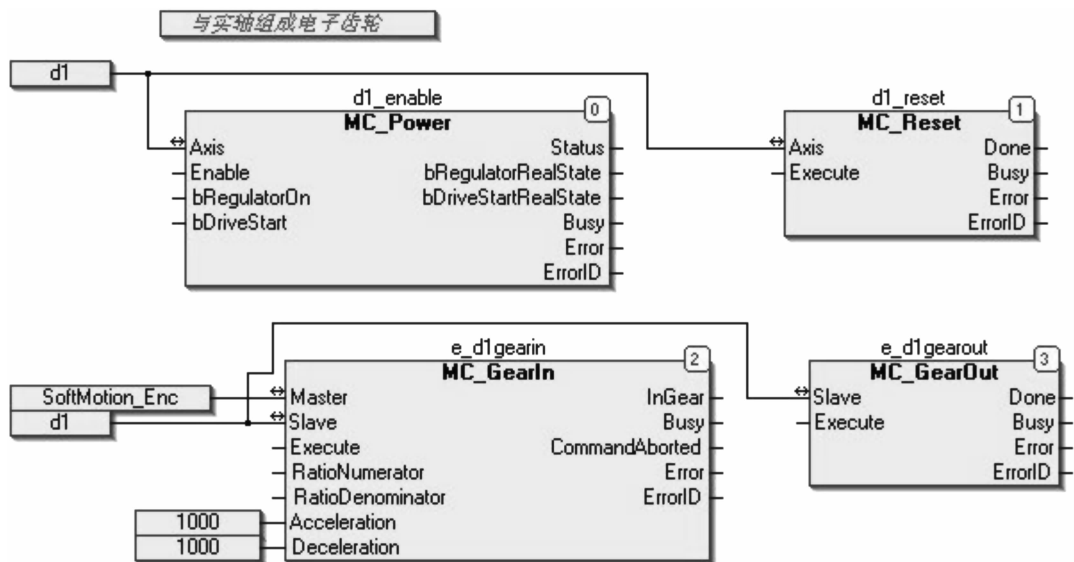


图 15-9　与编码器组成电子齿轮

捕捉主轴编码器位置，捕捉输入点为 LMC058 上的 Cap0（I1. 6）、Cap1（I3. 6），如图 15-10 所示。

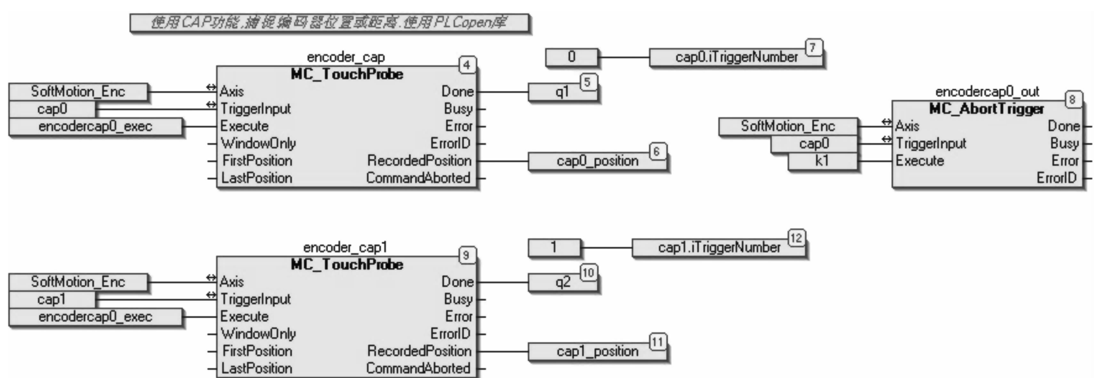


图 15-10　主轴编码器位置的捕捉

Motion Encoder 可以组态在主轴编码器口，也可以组态在专家的 I/O 口，区别是 I/O 口上只能接入增量编码器。

读主轴编码器数据如图 15-11 所示。

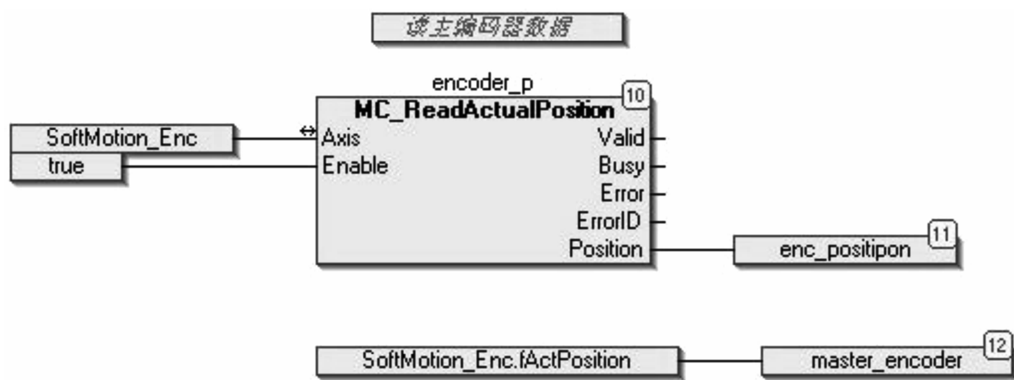


图 15-11　读主轴编码器数据

读主轴编码器数据时，有两种数据可读：一种是编码器定义的单位数据，如定义编码器转一圈为 360°，到 360°后自动清零，即模数模式。这样读到的数据在 0 ~ 360 之间。采用的方法是：功能块或结构变量的浮点实际位置；另一种数据是编码器实际数据，单位是脉冲数，是一个有符号的整数 DINT。例如编码器一转的脉冲是 4096，则转两圈就是 8192。它随着正传而增加，反转而减少。编码器的两种单位如图 15-12 所示。

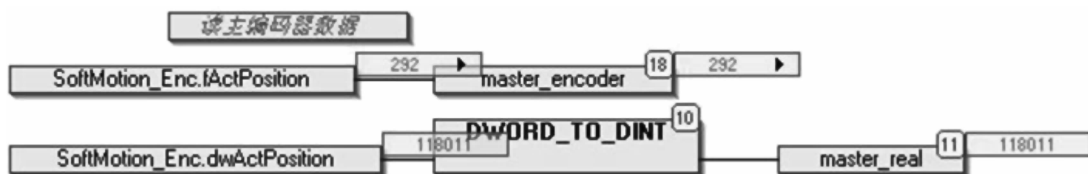


图 15-12　编码器的两种单位

2. Standard Encoder 标准编码器组态编程

点“DM72F0”按鼠标右键，添加设备。选择“Standard Encoder”，双击“Standard Encoder”打开配置画面，按需求配置即可。标准编码器组态如图 15-13 所示。



图 15-13　标准编码器组态

配置好 Standard Encoder 硬件组态后，就可以建立程序，进行编程应用。

3. Standard Encoder 应用编程

问题：Standard Encoder 能否组成电子齿轮功能？

在程序中，用 Standard Encoder 建立一个电子齿轮功能，如图 15-14 所示。

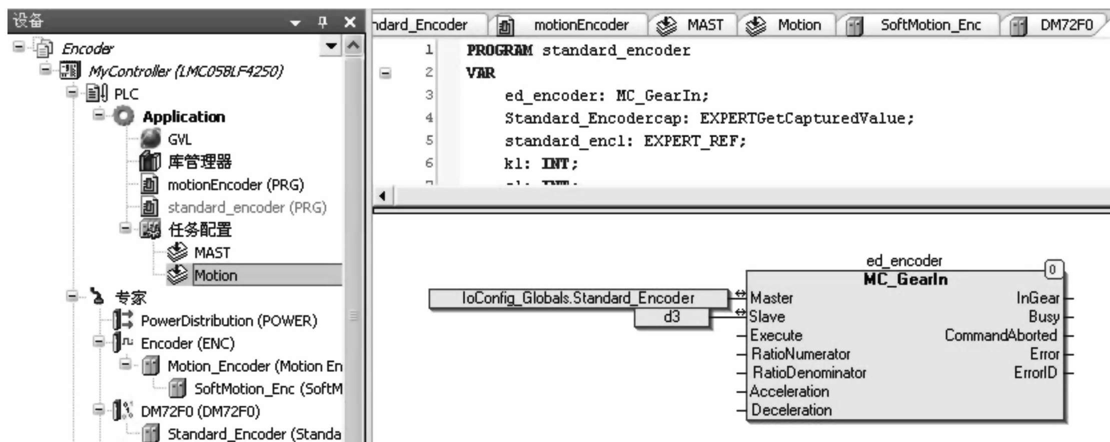


图 15-14　建立电子齿轮

无论把程序放在 MAST 还是 Motion 编译时都出错，如图 15-15 所示。

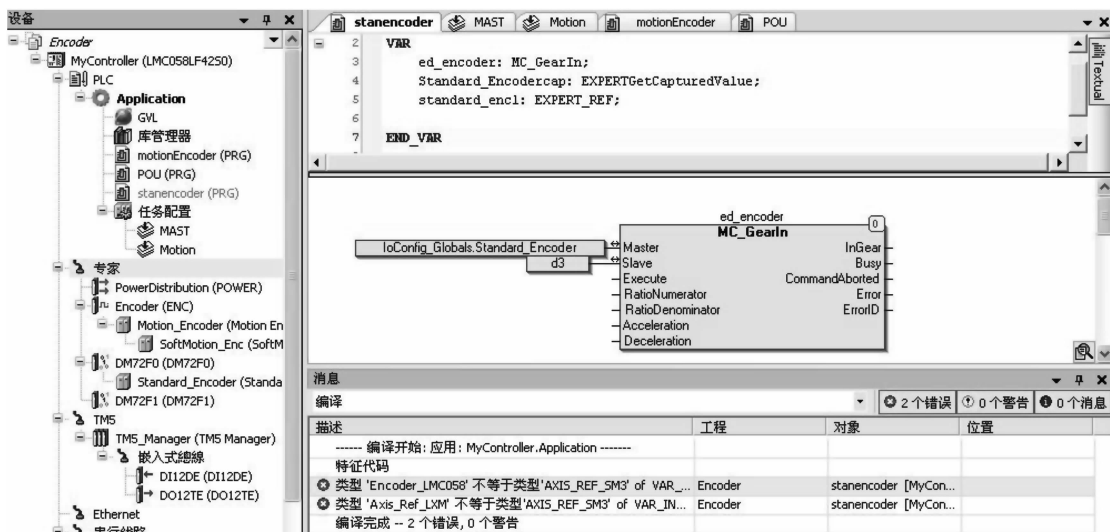


图 15-15　编译出错图示

取消电子齿轮功能，建立标准编码器功能块。编译结果图示如图 15-16 所示。

编译成功。说明 Standard Encoder 不能用于电子齿轮功能。只有 Motion Encoder 才能用于与其他轴的同步功能。

4. Standard Encoder 的数据读出

可以采用功能库 SEC_EXP 中的 Encoder 功能块 ENCODER_LMC058。



图 15-16　编译结果图示

这里要注意的是：功能块的名字不是随意定义的，它要和你组态好的编码器名字一致。这样程序块与实际编码器就关联起来。如图 15-17 所示，似乎名字很烦琐，不过不用担心，通过点击选择框，就可以看到组态好的编码器名字。

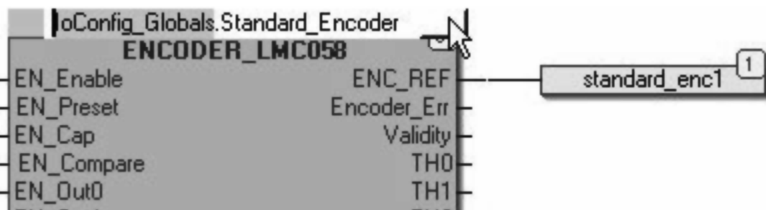


图 15-17　标准编码器功能块的命名

命名好功能块的名字后，就可以使用这个功能块了。首先看一下功能块的输入引脚。

功能块输入：

EN_Enable、EN_Preset、EN_Out0、EN_Out1 没有实际作用。

EN_Cap：如果是真，使能位置捕捉功能。

EN_Compare：如果是真，使能位置比较功能。

F_Enable：如果是真，使能功能块。

F_Preset：如果是真，计数复位到预置值。

F_Out0：如果是真，使能开关量输出 1。

F_Out1：如果是真，使能开关量输出 2。

ACK_Overflow：复位溢出标志。

ACK_Preset：复位预置标志。

ACK_Cap0：复位位置捕捉 1 完成标志。

ACK_Cap1：复位位置捕捉 2 完成标志。

SuspendCompare：如果为真，冻结比较结果。

功能块输出：

ENC_REF：定义编码器输出源，为以后使用编码器创建了应用名。

Encoder_Err：编码器错误提示。

Validity：如果为真，表明功能块正确运行。

TH0、TH1、TH2、TH3：设定的 4 个阈值，当编码器数值到达每一个阈值后，相关点为真。

Overflow_Flag：溢出标志。

Preset_Flag：预置标志。

Cap0_Flag：位置捕捉 1 完成标志。

Cap1_Flag：位置捕捉 2 完成标志。

Reflex0：当编码器计数值小于、大于或等于阈值时，这个点为真。计数与阈值关系基于组态。

Reflex1：当编码器计数值小于、大于或等于阈值时，这个点为真。计数与阈值关系基于组态。

Out0：开关量输出 1。

Out1：开关量输出 2。

Low_Limit：对增量编码下限的设定。

High_Limit：对增量编码上限的设定。

EncoderValue：编码器计数值。

知道了这些引脚的定义，就可以使用这个功能块来读编码器数据，如图 15-18 所示。

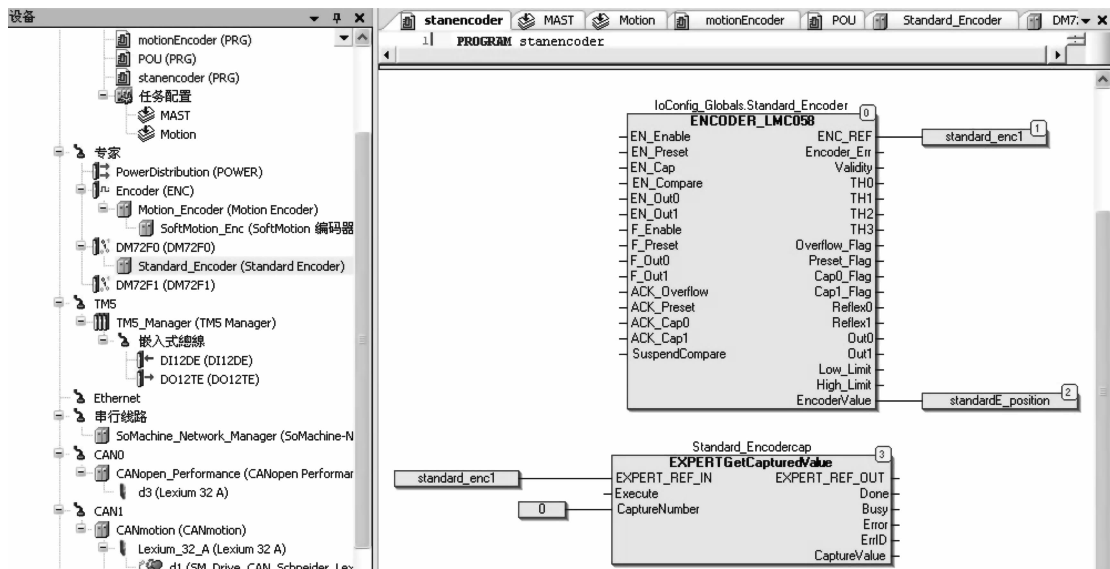


图 15-18　编码器数据的读取

使功能块有效，必须让 F_Enable 为真。

要使用捕捉功能，除了组态好捕捉输入点外，还要在功能块 ENCODER_LMC058 中，使能引脚 EN_Cap，使用捕捉功能块 EXPERTGetCapturedValue 来读出捕捉到的数据。

5. 对驱动器下电动机位置的捕捉

以上都是对主轴编码器位置的捕捉。在实际应用中也有对伺服轴电动机位置的捕捉，如图 15-19 所示。例中是对一个伺服轴 d3 位置的捕捉。

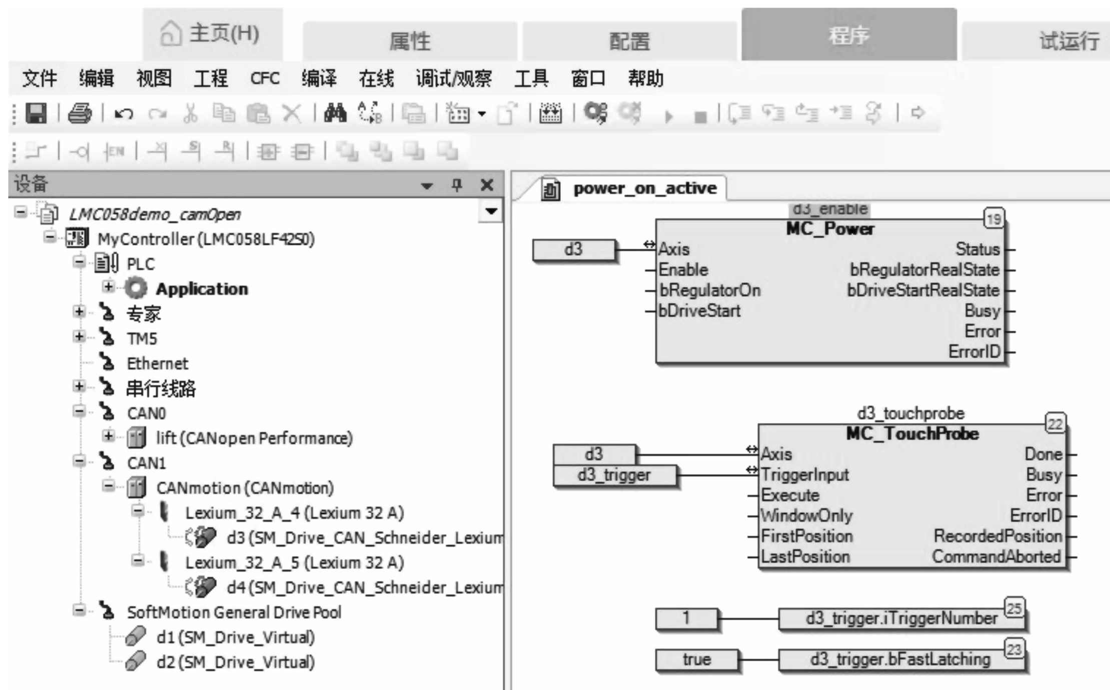


图 15-19　对电动机轴位置的捕捉

在功能块中，d3_trigger 是一个结构变量，如图 15-20 所示，它定义了触发捕捉位置的输入点。当 bFastLatching 为“false”假时，这个捕捉位置的触发可以采用控制器上的输入点，输入点地址或变量赋值给 binput，但捕捉的轴位置不是很精确，它基于运动任务的扫描时间。当 bFastLatching 为“ture”真时，这个捕捉位置的触发可以采用伺服驱动器上的输入点，这个输入点的定义要采用写控制字的办法来实现，而且捕捉的轴位置很精确。

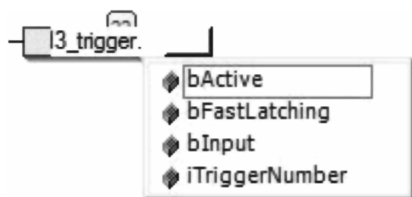


图 15-20　触发点的定义

通过查伺服驱动手册，我们可以知道启动位置捕捉的命令字。定义驱动器输入点参数如图 15-21 所示。

根据参数，驱动器的触发输入点定义需要 CANopen 总线写入的功能块，如图 15-22 所示。

如果驱动器连接在 CANmotion 总线下，则写入驱动器的功能块如图 15-23 所示。

这个功能块的 CAN 总线参数写入是对 CANmotion 总线下的伺服驱动器有效的。

启动位置捕获　通过参数 Cap1Activate，启动位置捕获。

参数名称 HMI 菜单 HMI 名称	说明	单位 最小值 出厂设置 最大值	数据类型 读 / 写 持续 专业	通过现场总线的参数地址
Cap1Activate	捕捉输入 1 启动 / 停止 **0 / Capture Stop**：中断捕捉功能 **1 / Capture Once**：启动一次性捕捉功能 **2 / Capture Continuous**：启动连续性捕捉功能 执行一次捕获时，将在捕获到第一个值时结束执行该函数 进行连续捕获时，将连续进行捕获。 变更的设置将被立即采用。	- 0 - 2	UINT16 UINT16 读 / 写 - -	CANopen 300A:4_h Modbus 2568

设置脉冲沿　通过参数 Cap1Config 可设置将在哪一个脉冲沿捕捉位置。

参数名称 HMI 菜单 HMI 名称	说明	单位 最小值 出厂设置 最大值	数据类型 读 / 写 持续 专业	通过现场总线的参数地址
Cap1Config	捕捉输入 1 的配置 **0 / Falling Edge**：下降沿时的位置捕获 **1 / Rising Edge**：上升沿时的位置捕获 变更的设置将被立即采用。	- 0 0 1	UINT16 UINT16 读 / 写 - -	CANopen 300A:2_h Modbus 2564

图 15-21　定义驱动器输入点参数

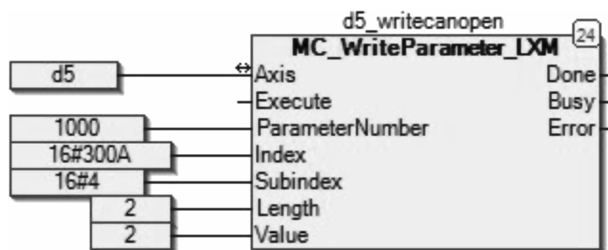


图 15-22　通过 CANopen 总线写入驱动器

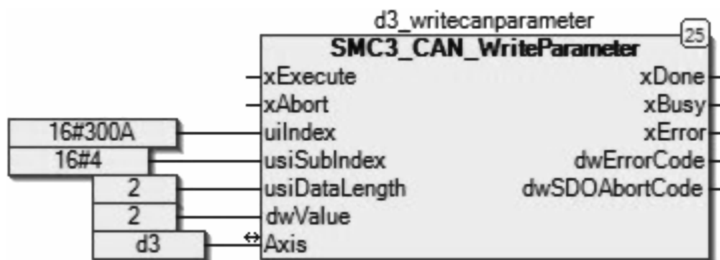


图 15-23　通过 CANmotion 总线写入对驱动器输入点的定义

第 16 章　基于以太网的数据交换

在 SoMachine 控制平台上，很多控制器带有以太网接口。我们可以通过以太网把编辑好的程序和硬件组态下载到控制器中，由于以太网传输速度快，所以对修改编辑程序的速度也会大大加快，更加方便调试程序。有了以太网口，也方便了远程对控制器主页的访问。当系统中有多个控制器时，通过以太网络，也可以实现资源的共享。在这一章，我们就来阐述相关的应用。

16.1　以太网组态

首先让我们看看以太网的组态。在 SoMachine 设备栏目下找到以太网（Ethernet），双击 Ethernet，打开 Ethernet 配置画面，如图 16-1 所示。在配置画面上，填写 Interface Name 和 Network Name 以及选择固定 IP 地址 fixed IP Address，在此栏目下，填写 IP 地址和子网掩码 Subnet Mask。如果系统中有多个控制器，分别设定即可。

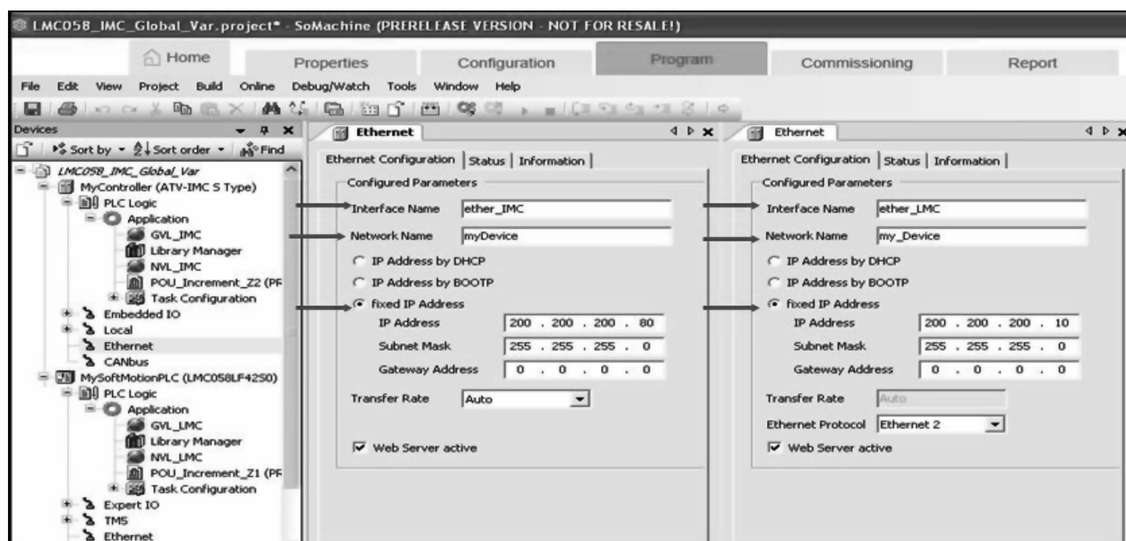


图 16-1　系统中控制器以太网的组态

设置好 SoMachine 的 IP 地址并下载后，就可以设置计算机网口的 IP 地址，例如，设置为如图 16-2 所示。

确认后，我们就可以在控制器的通信设定栏目下，扫描到连接在计算机上的控制器，如图 16-3 所示。

然后激活通信路径。这样就可以在 SoMachine 条件下，通过以太网连接到控制器，对控制器进行编程、组态、程序下载。也可以把计算机中的文件传输到控制器里，如图 16-4 所示。

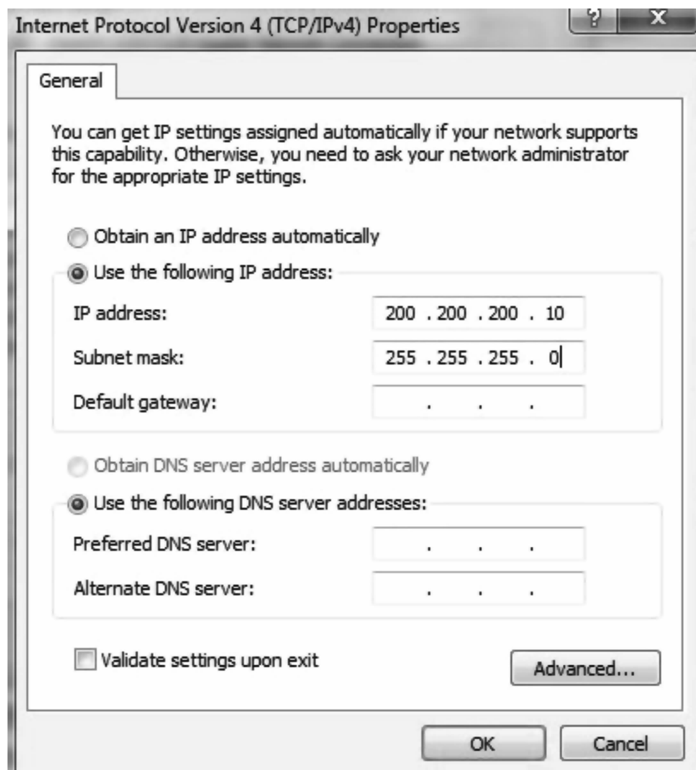


图 16-2　计算机网口 IP 地址的设定

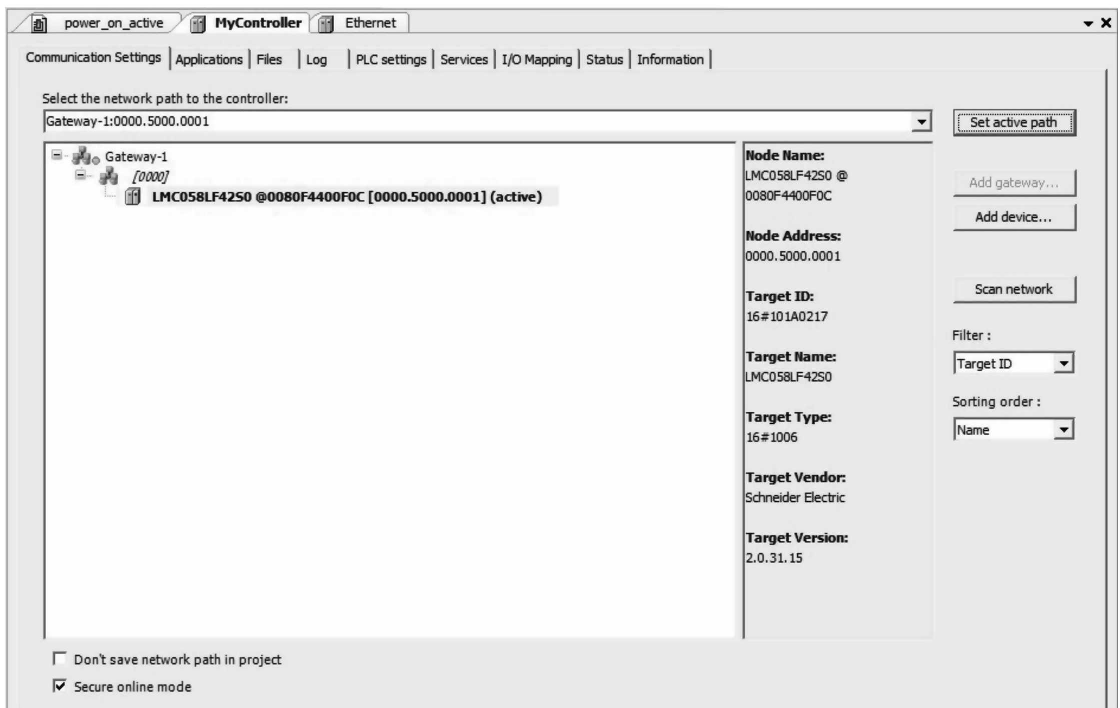


图 16-3　扫描到连接在计算机上的控制器

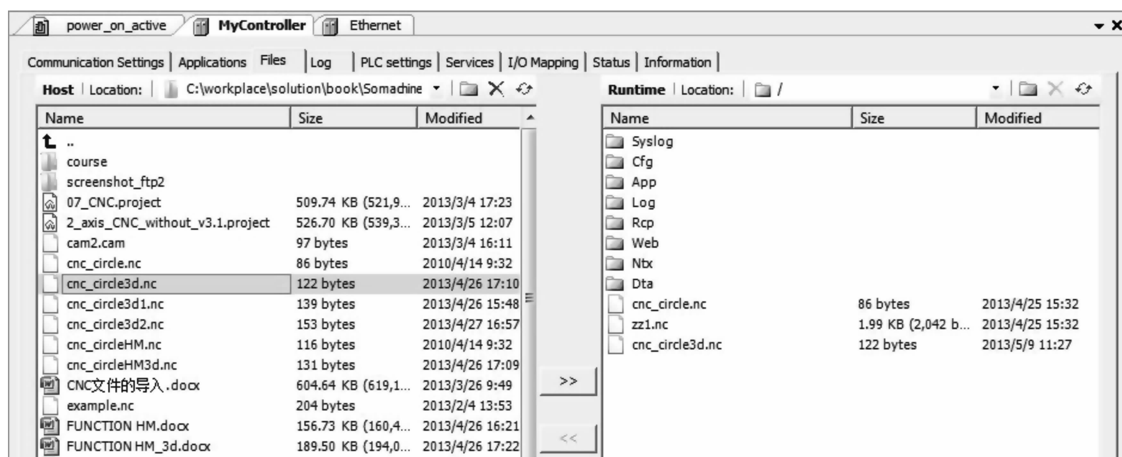


图 16-4　计算机和驱动器之间的文件传输

当然这种文件的传输是通过 SoMachine 进行的。在实际的应用中，有些客户没有安装 SoMachine，也想通过以太网访问控制器，那么，以下的一些方法就可以满足这种需求。

16.2　以太网服务 FTP 文本传输协议

FTP（File Transfer Protocol）是用于 internet 上的控制文件双向传输的协议。通过这个软件标准可以在 PC 和控制器之间传输文件，而无需安装 SoMachine 软件。例如，PC 向运动控制器发送数控 G 代码文件、凸轮曲线文件等。在用计算机直接连接控制器时，要注意的是控制器有初始的用户名和密码如下：

M258，LMC058，XTBGC：　LOGIN：　USER　　Password：　USER

IMC：　LOGIN：　USER　Password：　USER

用 USB 线进入网页，地址是：90. 0. 0. 1。

以太网的设置如上节所述做好后，就可以登录控制器了。首先连接好以太网，打开浏览器，输入 IP 地址。需要注意的是：用网线连接时，有些计算机要把无限网关掉，才能进入控制器网页。打开网页后，首先出现的是如图 16-5 所示画面，要求输入登录密码。默认的密码是：USER。

登录后，网页会显示控制器名称和外观。在网页左上边会显示控制器在某一功能下的操作菜单。例如，在首页下，会显示首页下的操作项：语言。在左下边是目前控制器的信息和状态以及控制面板，可以操作控制器起动和停止。控制器网页首页如图 16-6 所示。

在首页控制器名称下，有 4 个功能项，分别是监视（monitoring）、诊断（Diagnostics）、维护（Maintenance）和设置（Setup）。在监视功能下，我们可以看到 PLC 情况，如系列号、版本号和组态情况。控制器信息如图 16-7 所示。

除了看到控制器信息，我们还可以看到扩展模板的信息。例如控制器本体带有的 I/O 点模块。预览扩展模块如图 16-8 所示。

在诊断功能下，我们可以诊断控制器工作状态、以太网工作状态等。诊断功能下的控制器状态如图 16-9 所示。

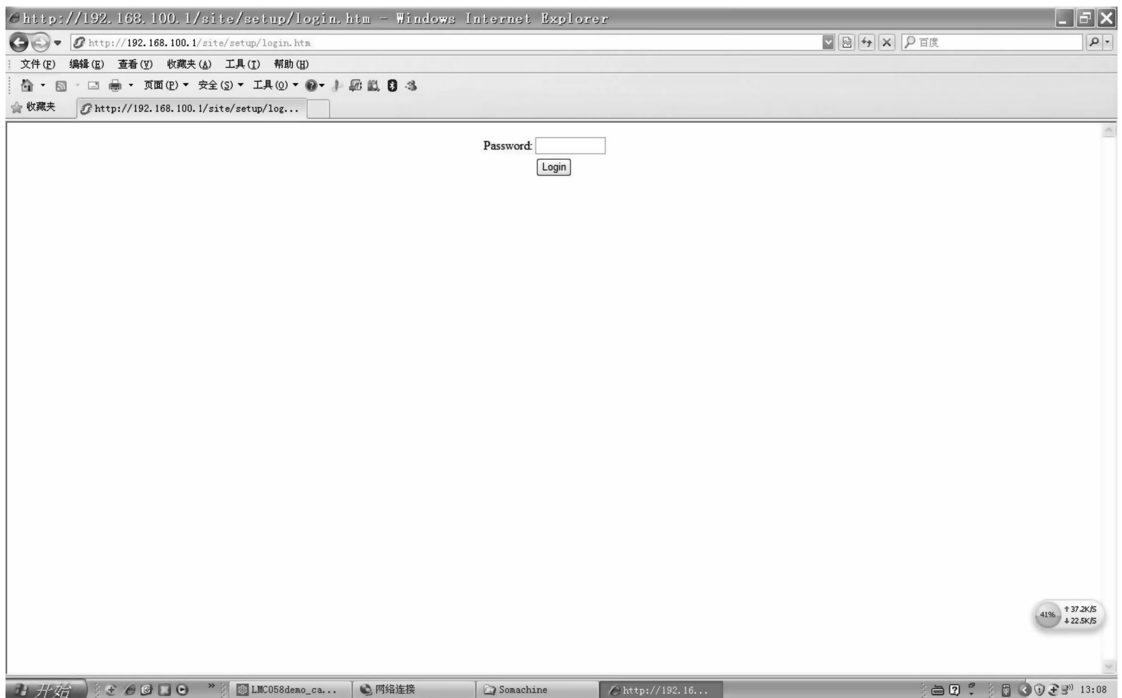


图 16-5　登录控制器网页



图 16-6　控制器网页首页

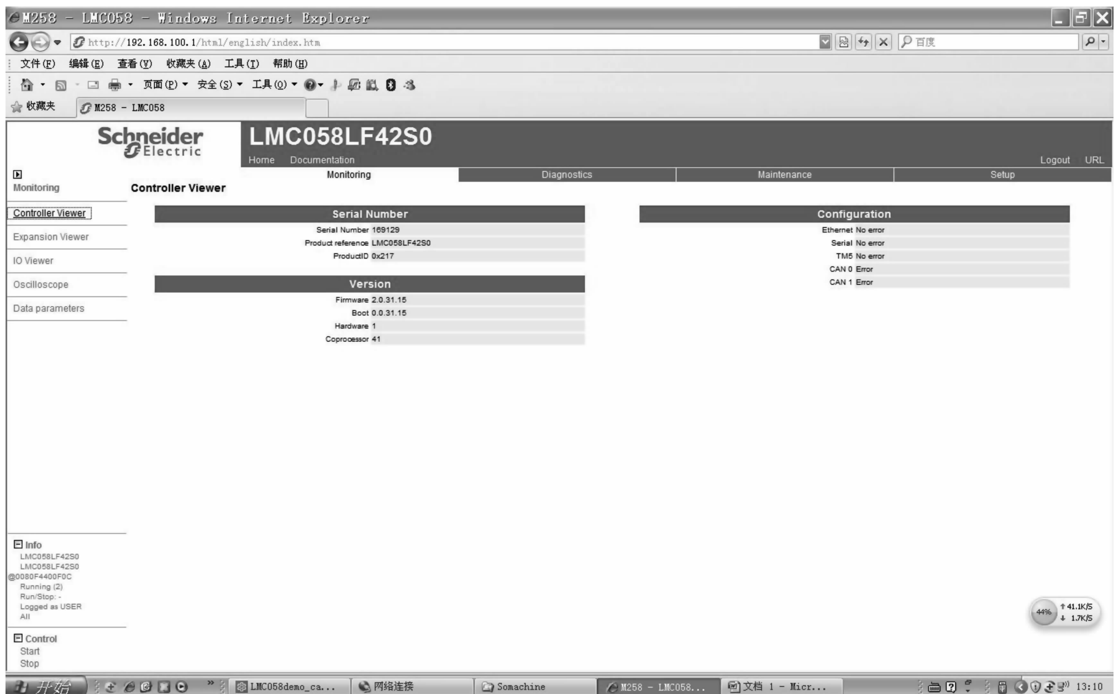


图 16-7　控制器信息

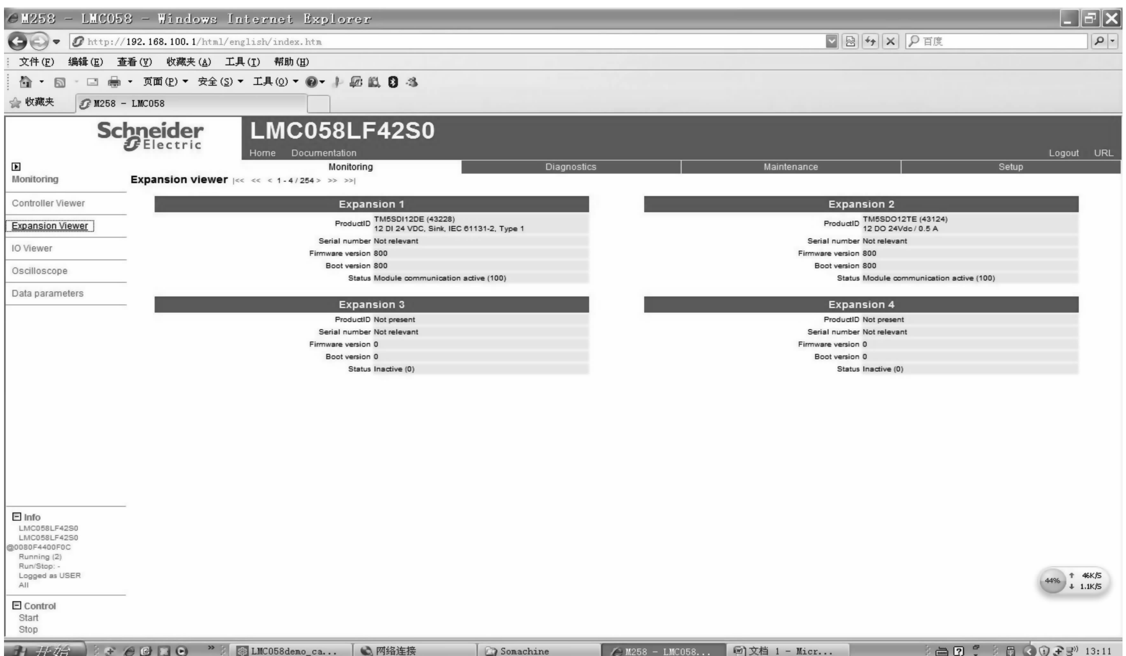


图 16-8　预览扩展模块

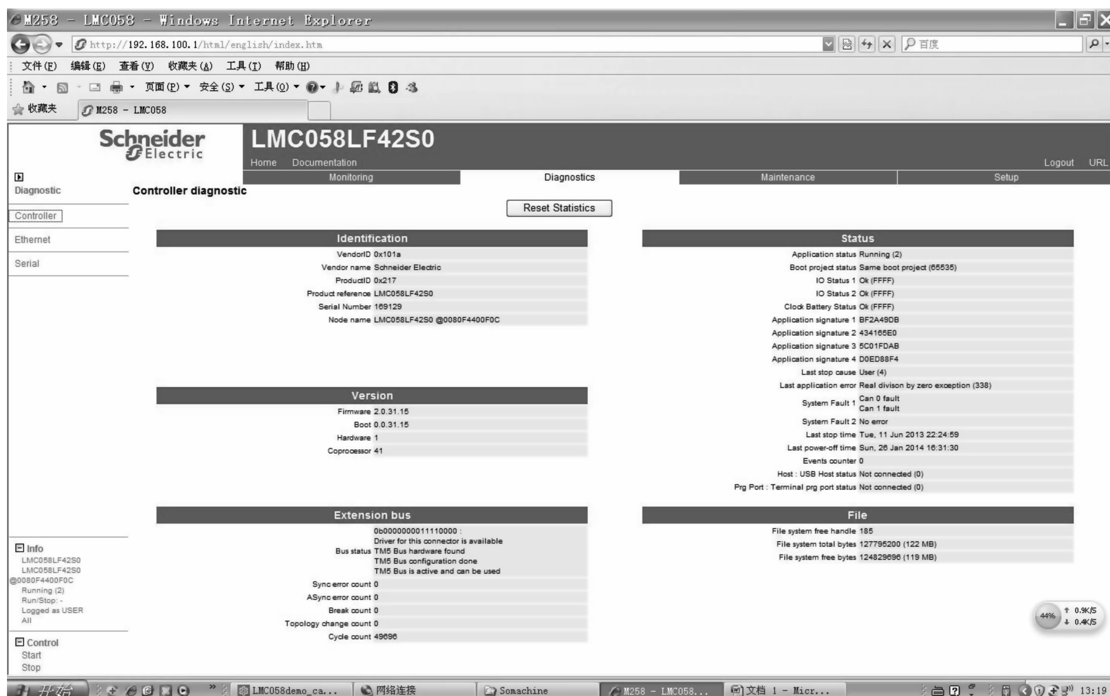


图 16-9　诊断功能下的控制器状态

在设置功能下，可以修改登录网页的密码，如图 16-10 所示。

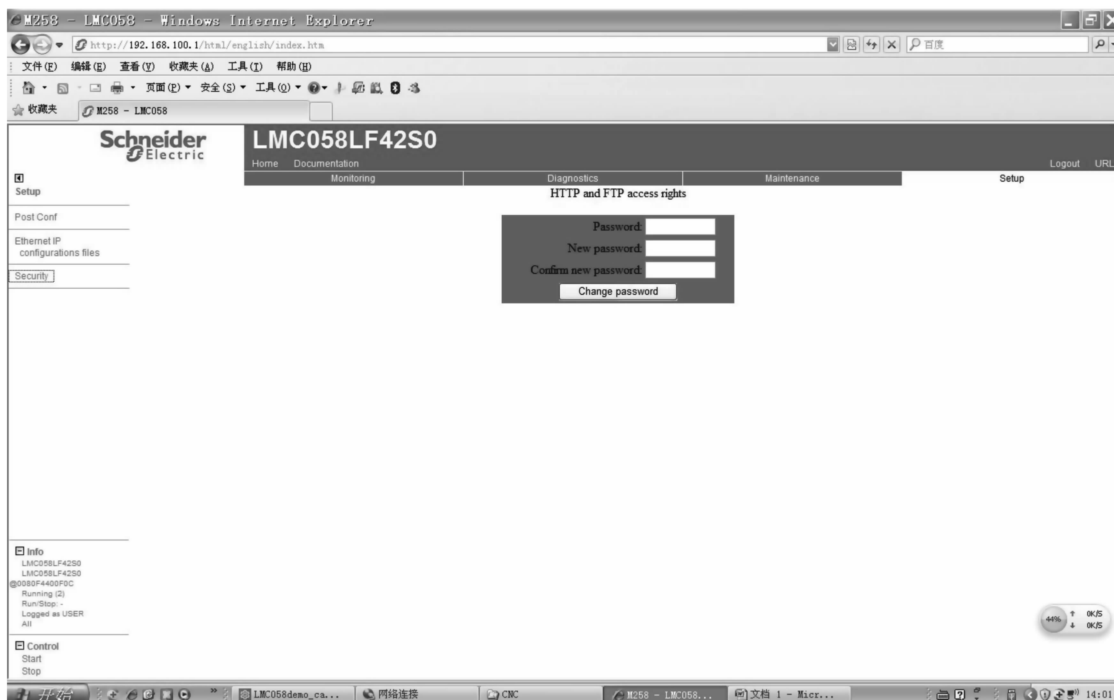


图 16-10　设置功能下修改登录密码

除了直接通过以太网访问控制器的主页外，我们还可以下载一些免费的 FTP 软件来向控制器内传送文件。例如，FileZilla。安装此软件并打开后，如图 16-11 所示。



图 16-11　打开软件界面

在主机处填上控制器的 IP 地址、用户名（USER）和密码（USER），并点击“快速连接”。连通后，可以看到读上来的控制器软件路径，如图 16-12 所示。



图 16-12　远程站点情况

连通后在本地站点找到要传送的文件并双击此文件，这个文件就被发送到控制器内的用户文件夹内，如图 16-13 所示。



图 16-13　把计算机内的 G 代码文件发送到控制器内

16.3　SoMachine 协议

在一个系统中，我们有时会连接很多控制器，这样就要求控制器之间能够资源共享。在一个控制器中的若干变量可以方便地为另一个控制器所用。在施耐德电气系统中，我们设计了一个 SoMachine 协议来完成这个功能，它的结构如图 16-14 所示。

在组成这个共享系统中，我们要传输的数据变量必须定义为全局变量，同时要遵循如下规定：

1）定义系统中的发送者/接收者；

2）必须定义在固定的变量列表中；

3）这些列表分别在发送者和接收者一端；

4）数据采取广播的方式传送；

5）单向传输数据。

首先，我们在发送者的应用 Application 栏目下添加全局变量列表，并且命名发送者名称，如图 16-15 所示。

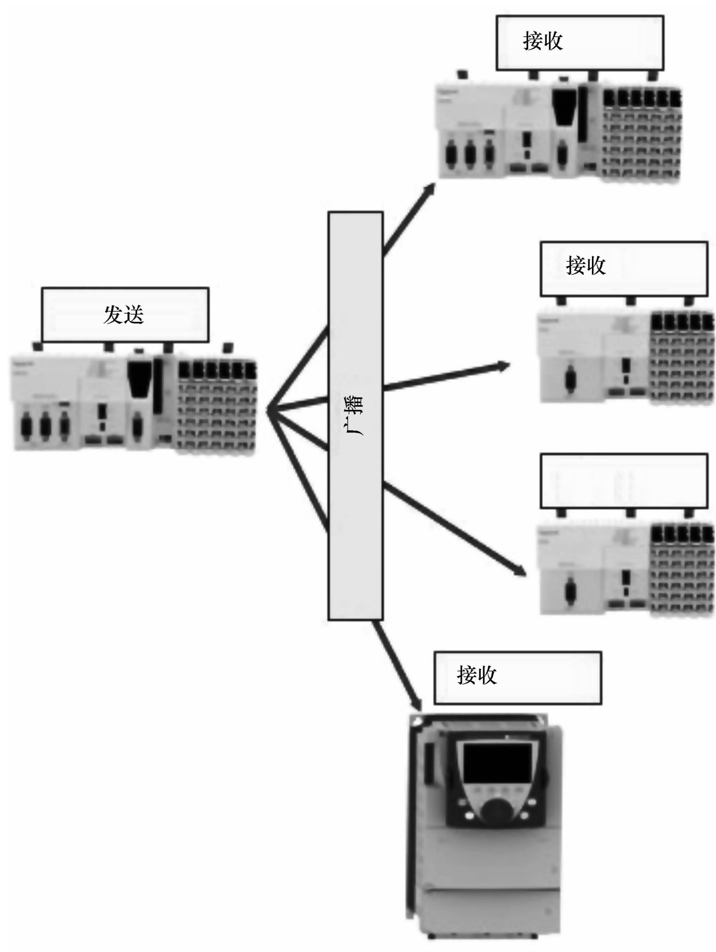


图 16-14　资源共享结构

然后，在建立好的发送者全局变量下拉菜单下，选择网络变量发送者属性，如图 16-16 所示。

打开网络变量发送者属性画面，定义发送者属性，如图 16-17 所示。

选择网络类型（Network type）为 UDP。

选择同步交换数据的任务：MAST。

激活传送时间（Cyclictransmission）并定义时间周期。

确认后，发送者图标变为 

。

定义完发送者，我们选定一个接收者，在接收者的应用栏目下添加全局网络变量列表。如图 16-18 所示，在这里命名接收者全局变量列表，选择同步周期任务和接收哪一个发送者的信息。

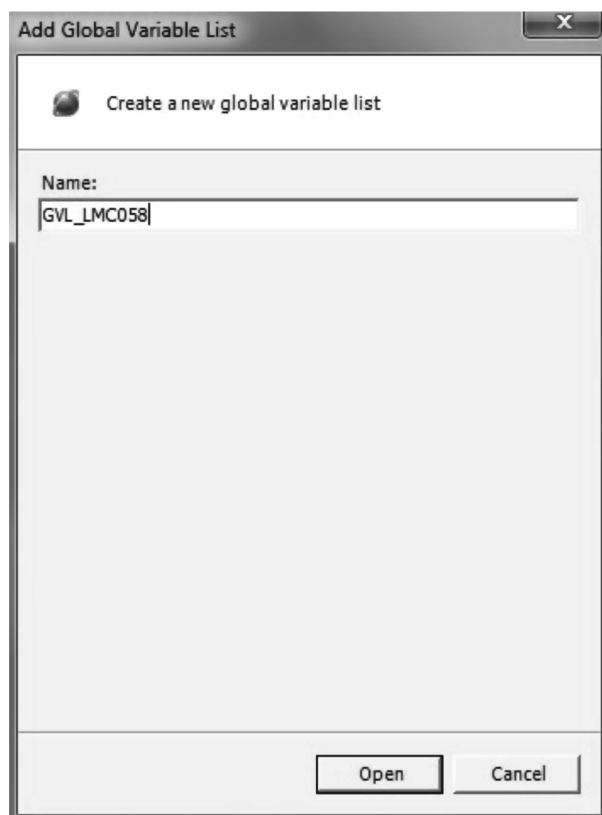


图 16-15　添加全局变量

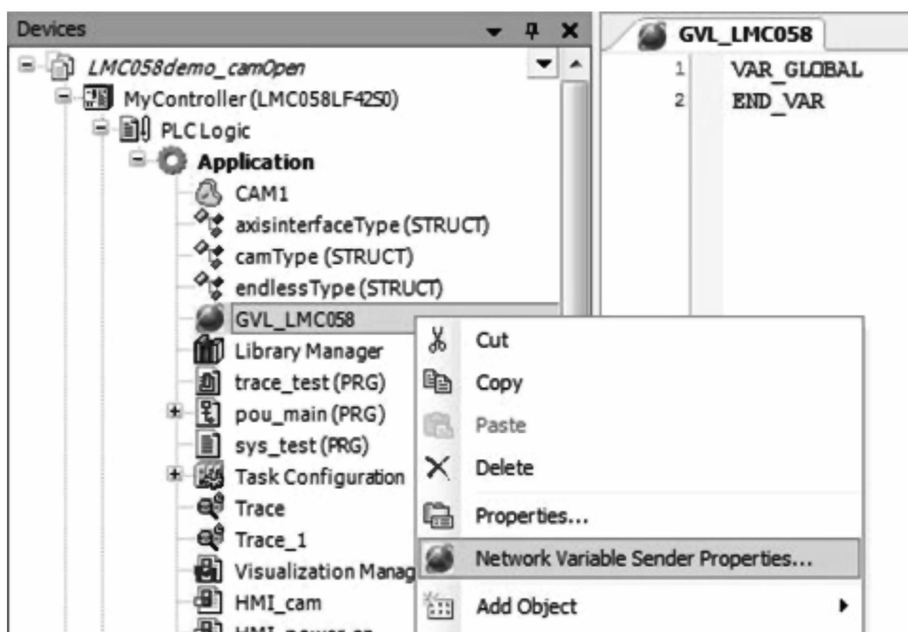


图 16-16　选择网络变量发送者属性

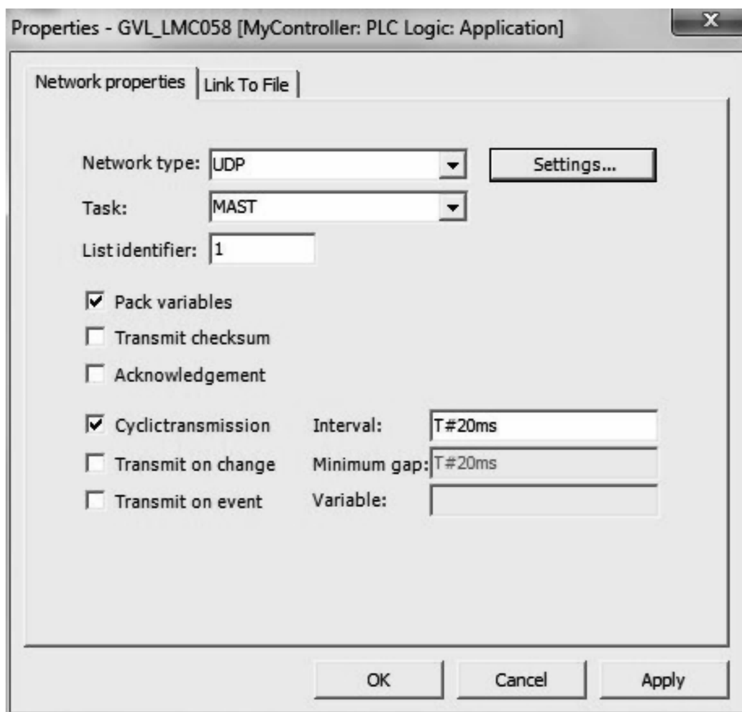


图 16-17　定义发送者属性

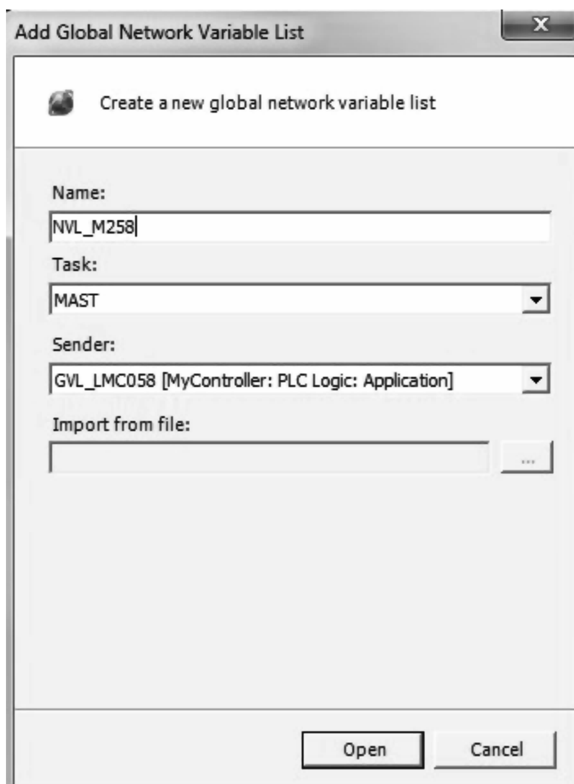


图 16-18　接收者全局变量列表

确认后，接收者网络全局变量列表如图 16-19 所示。

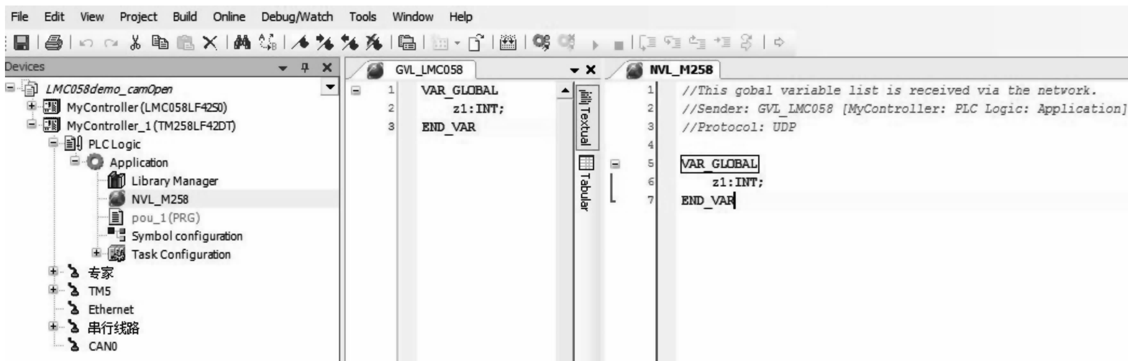


图 16-19　接收者网络全局变量列表

接收者网络全局变量列表建立好后，发送者变量列表中的变量自动在接收者变量列表中生成。

同样，也可以在接收者设备上建立一个发送者，在发送者设备上建立一个接收者，如图 16-20 所示。

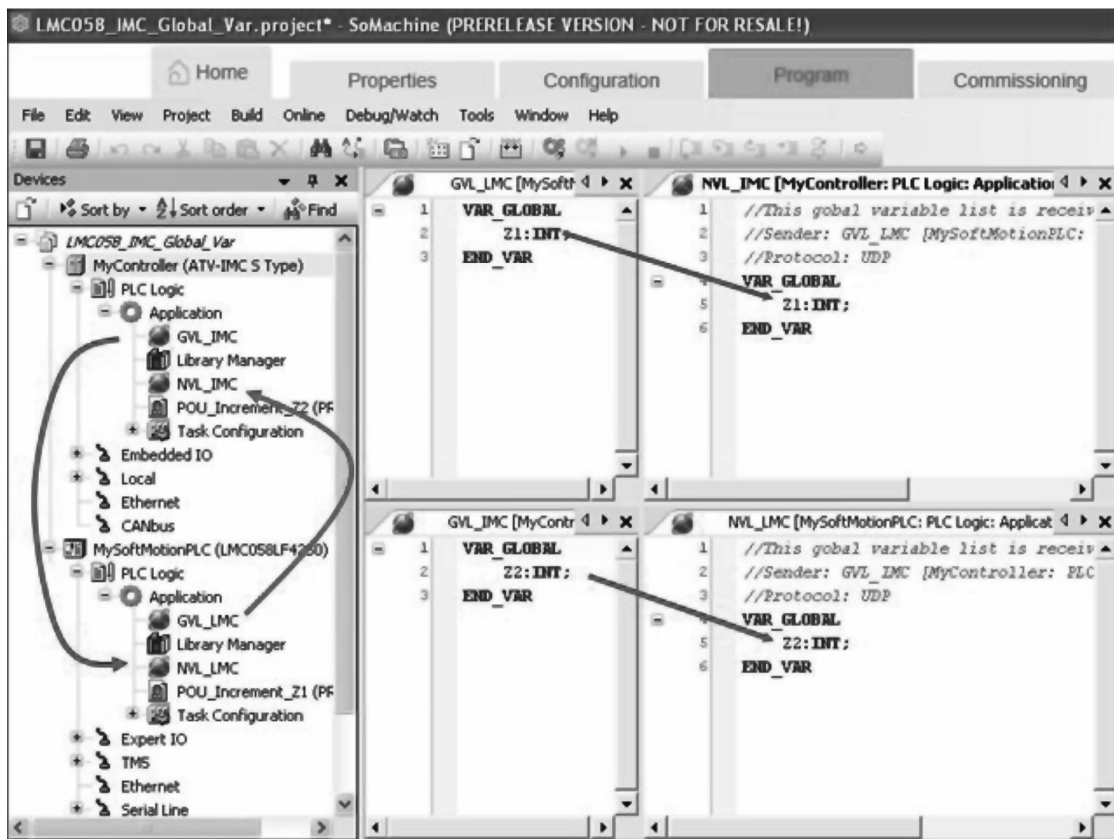


图 16-20　在 2 个设备上建立发送/接收者

通过建立这种网络结构，我们就可以实现变量共享，而且所有类型的变量都可以，包括结构变量。

16.4　客户/服务器模式

在上一节介绍的交互式共享系统中，变量的数据交换基本是 2 个控制器之间的交换并且周期地同步刷新交换数据，即交换数据是实时进行的。这一节我们介绍随机进行数据交换的方法，即基于 ModbusTCP 客户（Client）/服务器（Server）模式的数据交换。

1. Client/Server 客户/服务器模式

Client 和 Server 常常分别处在相距很远的两台控制器上，Client 程序的任务是将用户的要求提交给 Server 程序，再将 Server 程序返回的结果以特定的形式显示给用户；Server 程序的任务是接收客户程序提出的服务请求，进行相应的处理，再将结果返回给客户程序。

2. Client/Server 结构

即大家熟知的客户机和服务器结构。它是软件系统体系结构，通过它可以充分利用两端硬件环境的优势，将任务合理分配到 Client 端和 Server 端来实现，降低了系统的通信开销。施耐德电气的以太网系统 TCP/IP 基于这种客户机/服务器结构。它也是由客户端向服务器发出请求，然后服务器执行请求并把存储器内的数据进行交换。SoMachine 控制器即可以是服务器，也可以是客户机。而内置以太网口的控制器作服务器无需任何组态，而控制器的内置以太网口默认的通信端口为“3”。

组态时，我们在系统中选择一个设备作为服务器，例如控制器 IMC。我们开始服务器的组态。

（1）服务器的组态 Server-Project

在服务器 MyController1 端，我们建立一个全局变量，并且建立一个程序组织单元 POU。然后组态这个设备的以太网地址。服务器的组态如图 16-21 所示。

组态完服务器，我们开始客户机的组态。

Modbus TCP 客户端支持以下功能块：

- ADDM；
- READ_VAR；
- SEND_RECV_MSG；
- SINGLE_WRITE；
- WRITE_READ_VAR；
- WRITE_VAR。

注意：IMC 只应答 ModBus 地址 252。

（2）客户机组态 Client-Project

选定一个设备作为客户机，例如 MyController。在客户端建立一个程序组织单元 POU，利用上述的功能块，编写对服务器的读/写程序。客户端的组态如图 16-22 所示。

在客户端的程序中，我们用 ADDM 功能块定义服务器端的地址‘3 {200. 200. 200. 2} 252’。读/写功能块的 ObjType 为“0”即对象类型为%MW。FirstObj 为 102，即从%MW102



图 16-21　服务器的组态

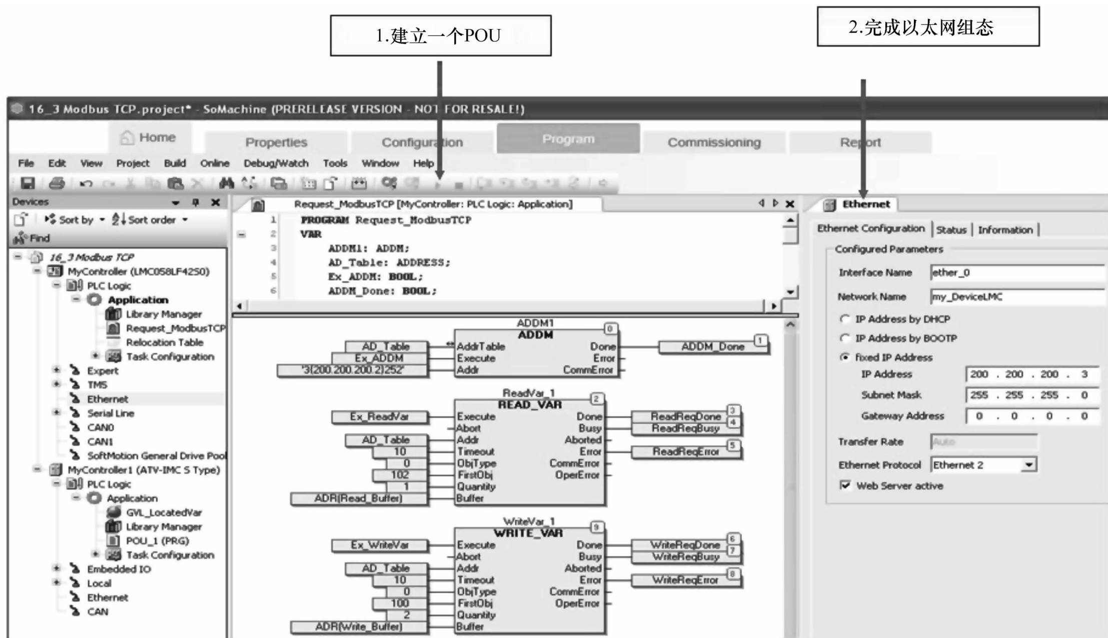


图 16-22　客户端的组态

开始读取。编写好程序，再把客户端设备的以太网地址配置好，这样就可以从客户端向服务器发送数据交换请求了。有关功能块的说明也可参见在线帮助。

16.5 连接无处不在

当今移动互联网的迅猛发展，像打开了无限连接的潘多拉盒子，使人们的连通、交流更加便利。也颠覆了传统的行为模式。这种潮流也带动了自动控制的发展，使信息的获得更加借助公共资源，使人们在监控设备、调试设备上不再受到地域、空间、时间、连线的限制。使对设备的连接无处不在。

施耐德电气跟上了这种潮流，积极响应用户对移动互联越来越多的要求，推出了基于公共资源的移动终端（智能手机、iPad）来监视、调试设备的技术。使得用户对设备的监视、调试、操作不再需要计算机和电缆。他们操作设备可以在任何地点，任何时间进行。移动操控设备如图 16-23 所示。

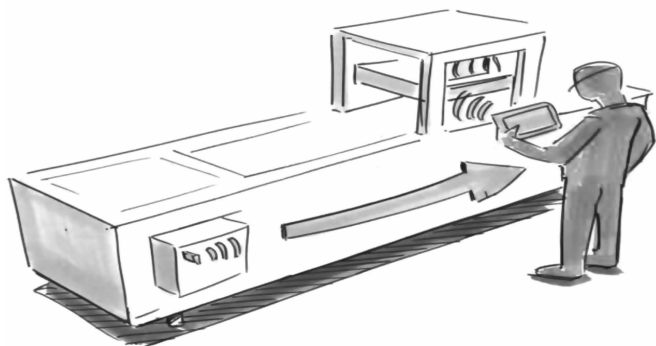


图 16-23　移动操控设备

如图 16-24 所示，通过移动终端，我们可以访问控制器的网页。通过 SoMachine 编辑的网页，我们就可以操控机器设备的运行了。



图 16-24　通过移动互联操控机器

我们首先采用 SoMachine 来编辑可视界面作为网页。可视界面的编辑，我们在第 9 章已经做了说明。编辑好的界面如图 16-25 所示。

与以往不同的是，我们在可视界面管理器下要建立一个网页可视界面。把要显示在网页上的可视界面名称填上，并且命名一个网页文件名，如图 16-26 所示。

在可视界面管理器里，你可以选择哪个界面可以显示在网页里，如图 16-27 所示。

设定好这些后，我们就可以定义控制器 IP 地址了，如图 16-28 所示。同时在 IP 地址下边的安全参数设置区，激活网页服务器。

然后，我们把编译成功的这些地址、界面及组态下载到控制器。把硬件结构搭好，无线互联如图 16-29 所示。

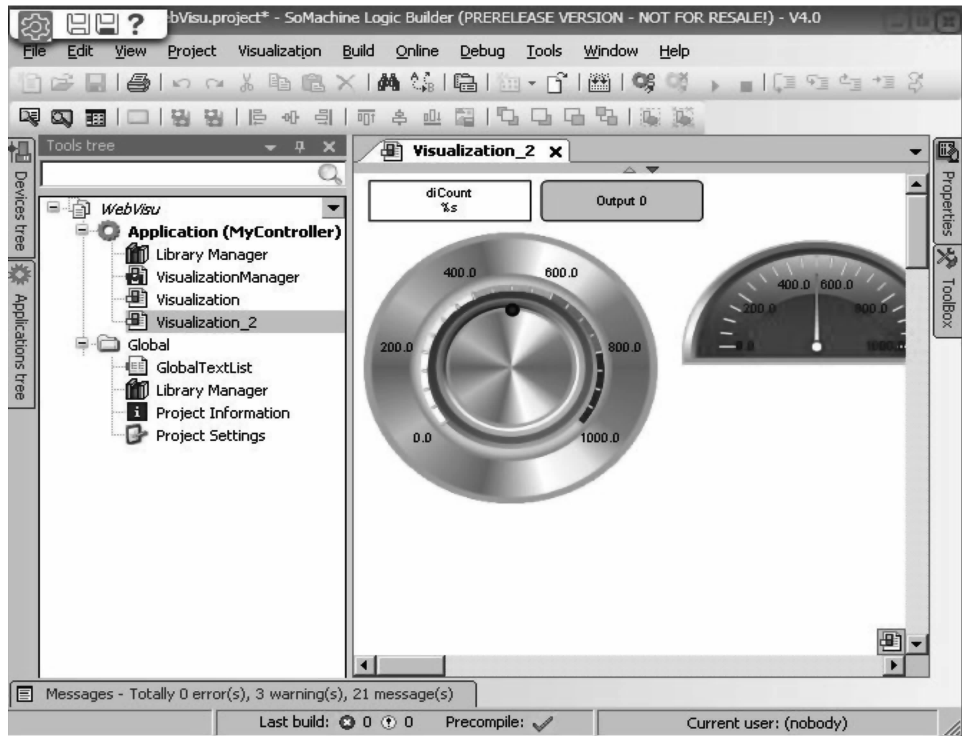


图 16-25　编辑好的可视界面

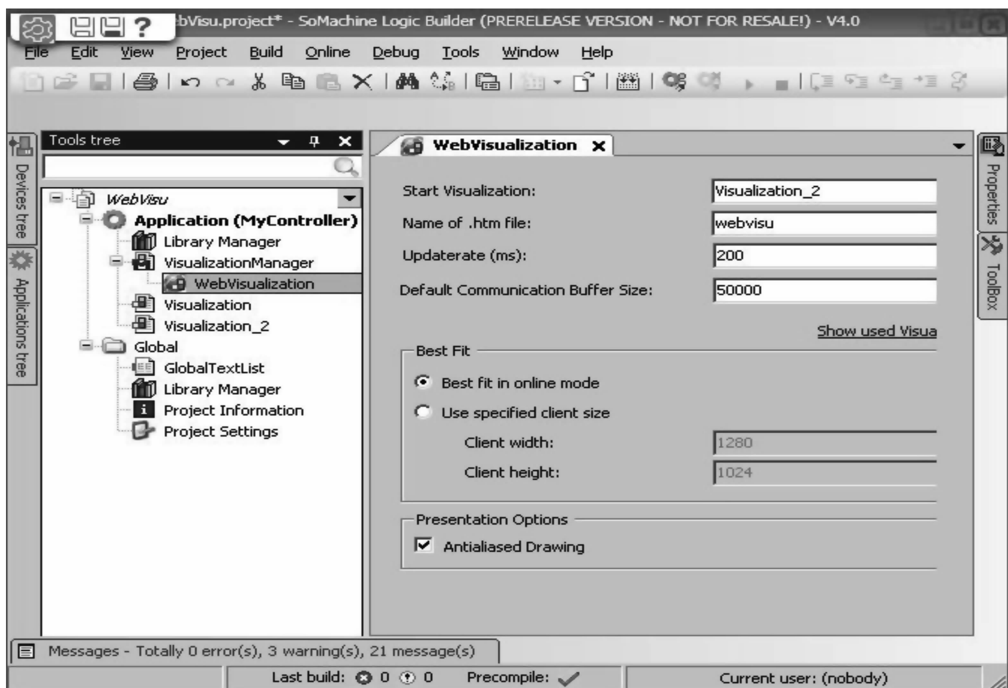


图 16-26　建立可视网页和网页文件名

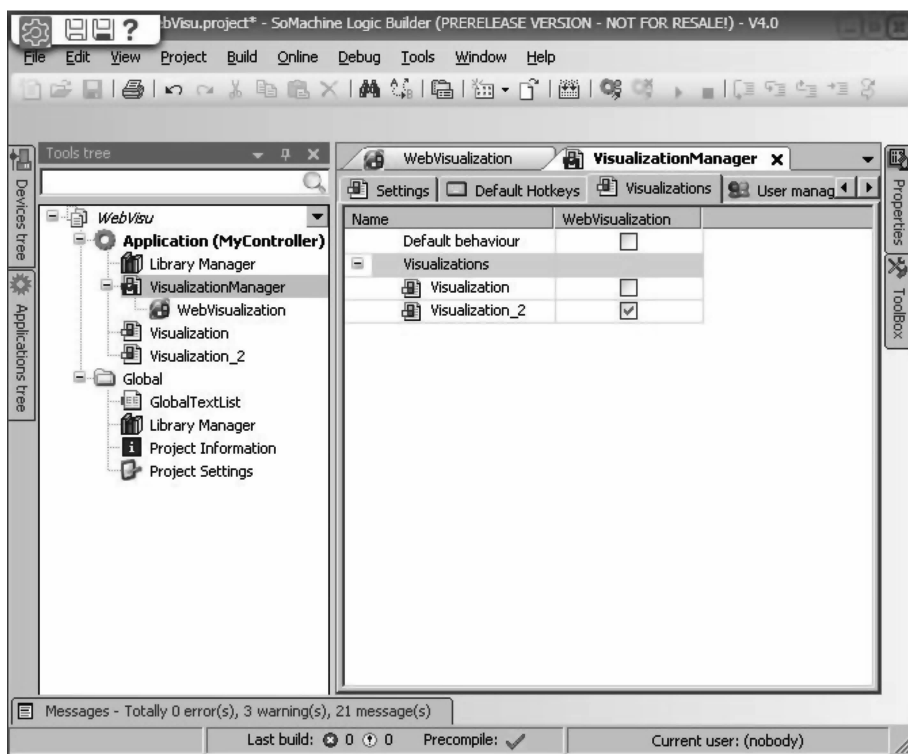


图 16-27　选择可在网页上显示的可视界面

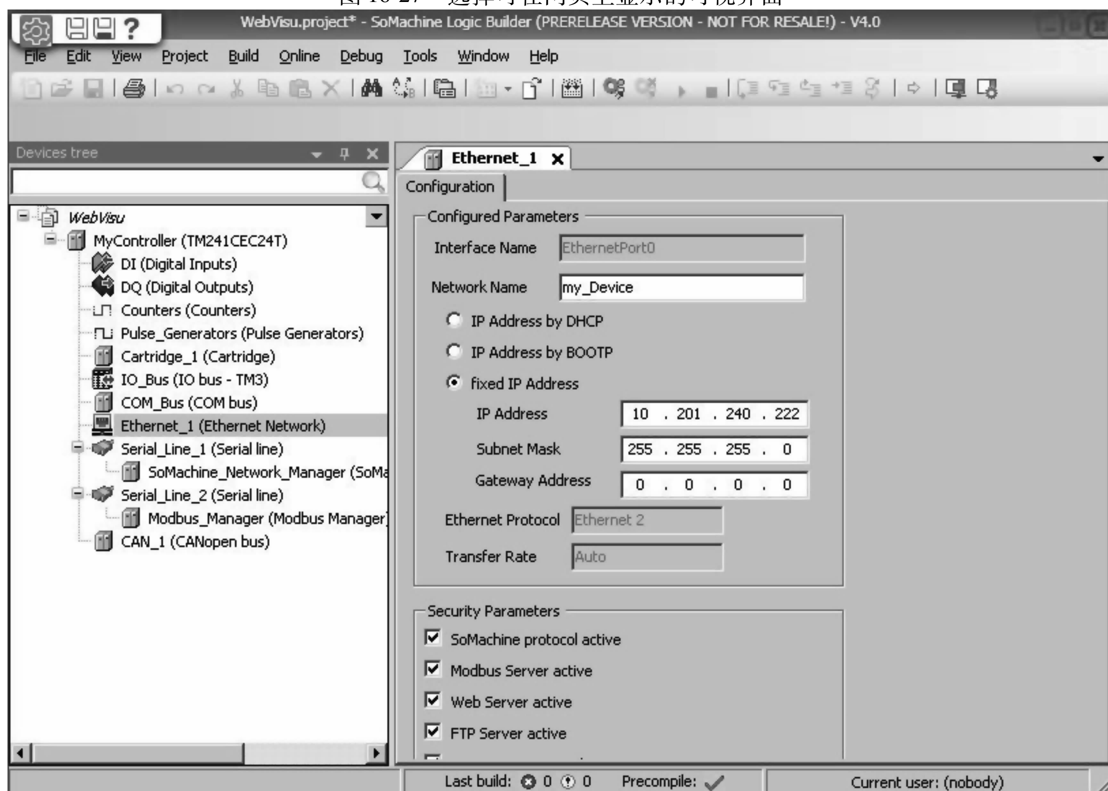


图 16-28　编辑控制器 IP 地址

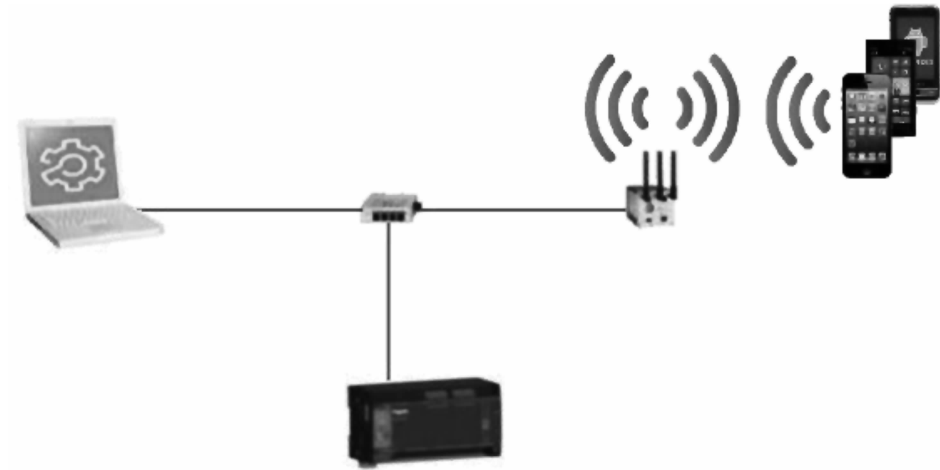


图 16-29　无线互联

通过任何移动终端的浏览器访问网页［IP 地址］:8080/［html 文件名］. htm

例如，我们示例中组态好的：10. 201. 240. 222:8080/webvisu. htm。

第 17 章　高速计数器

在对机器的自动控制应用中，对高速计数的需求是大量存在的，而且由于控制的日益复杂又繁衍出各种各样的计数要求。因此各个自动控制厂商都在自己的 PLC 中集成了强大的高速计数功能，使计数的频率不断增高，功能不断加强。SoMachine 控制器系列也是这样。M238 逻辑控制器的计数频率达到 100kHz，M258 逻辑控制器和 LMC058 运动控制器更是达到 200kHz。这使得控制器在控制各种机器时更加得心应手。

17.1　高速计数器简介

在 SoMachine 平台的所有的 M238/M258/LMC058 PLC 都包含高速计数输入，如图 17-1 所示。这些可以被组态为高速计数的脉冲来自：传感器、编码器、开关…。

高速计数器（High Speed Counter，HSC）是独立的扫描时间，因此它的速度不受 CPU 扫描时间的影响。

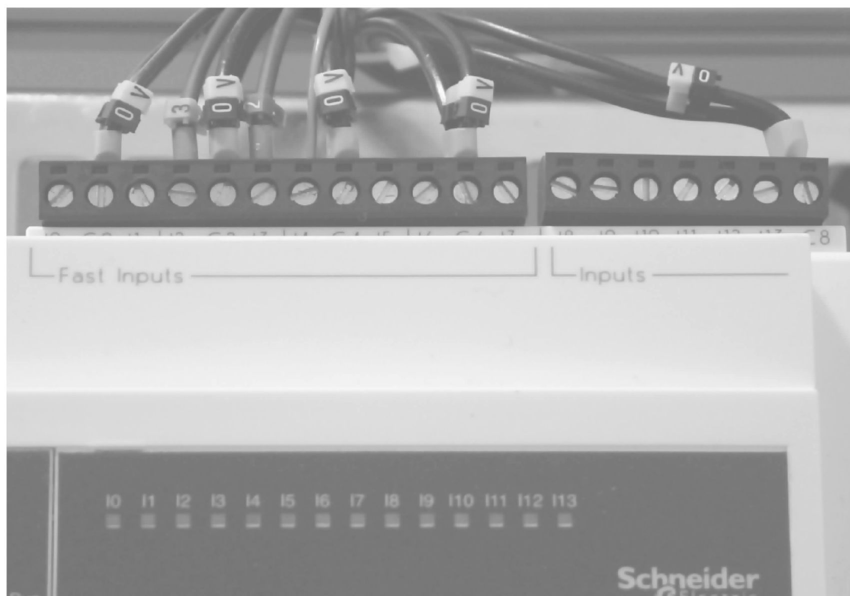


图 17-1　PLC 的高速计数输入

在控制器中的高速计数通常可以组态为两种类型：

简化（simple）：占用很少的输入点，功能有限；

主要（main）：更加复杂的应用，占用更多输入点。

简单计数-更少地占用输入点，功能有限。

●当到达设定值后，动作执行。

●动作执行完毕取决于主任务或快速任务的程序（不是事件任务）。

● 2 个简单的计数模式：

●一次性（One-shot）：上升/下降计数置位预设定值，然后等待 sync 信号来再次启动。

●模数回路（Modulo-loop）：上升/下降计数直到达到设定值。下一个脉冲，复位计数器并且启动计数。

选择简要计数如图 17-2 所示。



图 17-2　选择简要计数

主要计数-更加复杂的应用，更多地占用输入点。

●可以触发 2 个映射输出，输出点动作基于阈值的组态。

●映射输出→标准输出。

● 5 个复杂的计数模式：一次性（One-shot）、模数回路（Modulo-loop）、自由-大型（Free-large）、事件（Event）和频率计（Frequency meter）

选择主要计数如图 17-3 所示。

在组态计数器时，我们把每个输入都称为一个通道。

不同的计数器类型采用不同的通道数量：

●简要计数器使用一路通道。

●主要计数器可能采用多达 4 路通道。

●必须使用 I0 或 I4 启动。

高速计数器通道组态，我们可以：

●把多达 8 通道组态为“简化计数器”；

●把其中 4 个通道组态为简化计数器把 1 个通道组态为“主要计数器”；

●把 2 个通道组态为“主要计数器”。

计数通道的组态资源如图 17-4 所示。

组态通道为简化计数器时，用的通道只有 1 个，即输入点 I0，如图 17-5 所示。

组态为主要计数器时，用到的通道很多。这时用到哪个通道就可以激活哪个通道。如图 17-16 所示，我们激活了很多通道。



图 17-3　选择主要计数

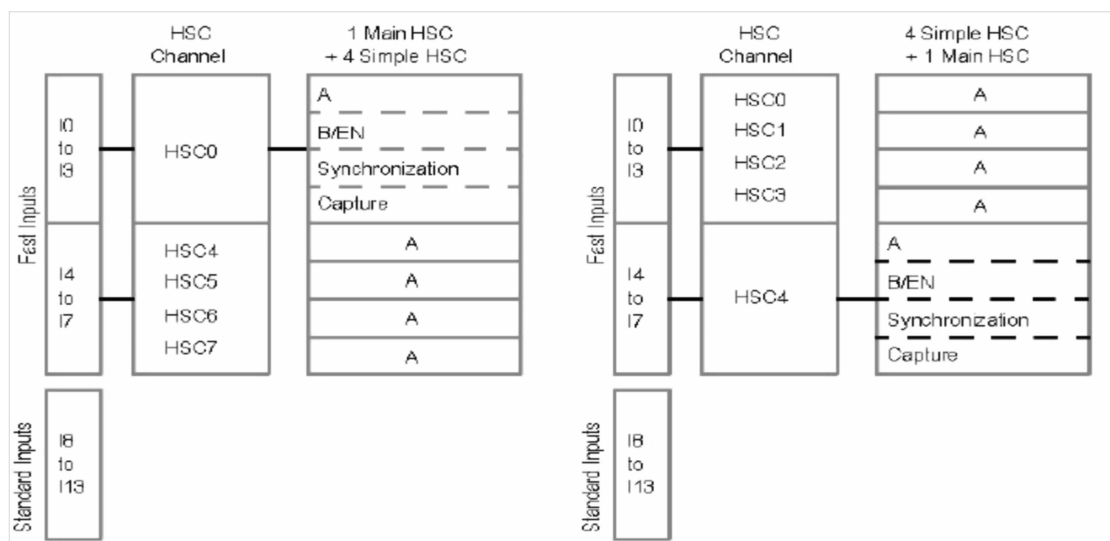


图 17-4　计数通道的组态资源

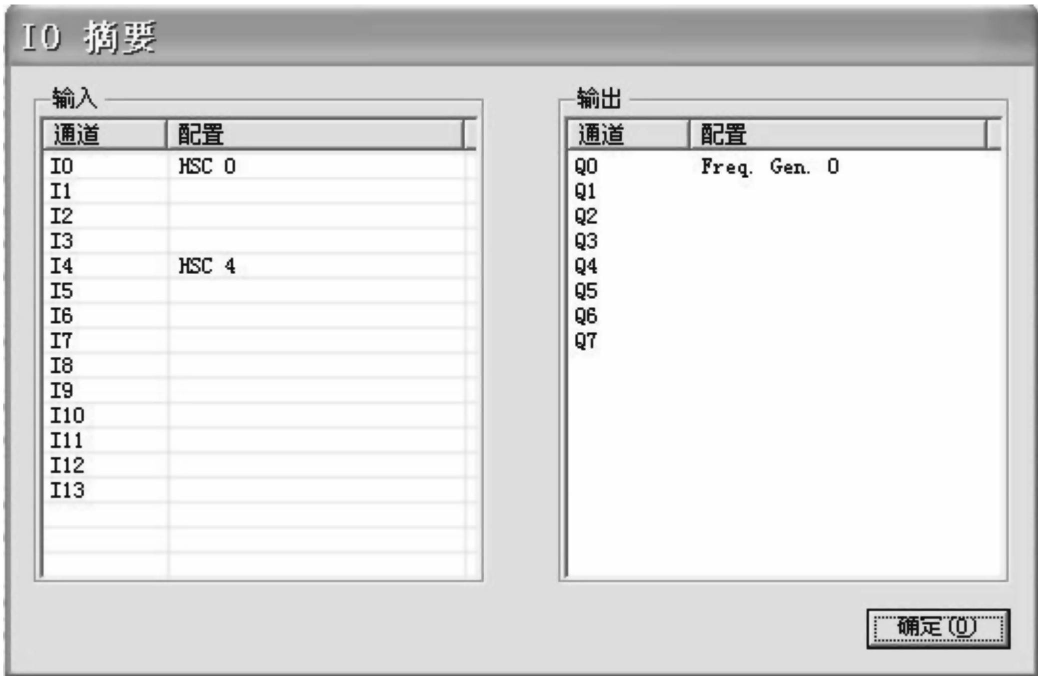


图 17-5　输入点摘要



图 17-6　在主要计数模式下启用多个通道

点击 IO 摘要，我们可以看到被激活的通道占用的输入点，如图 17-7 所示。

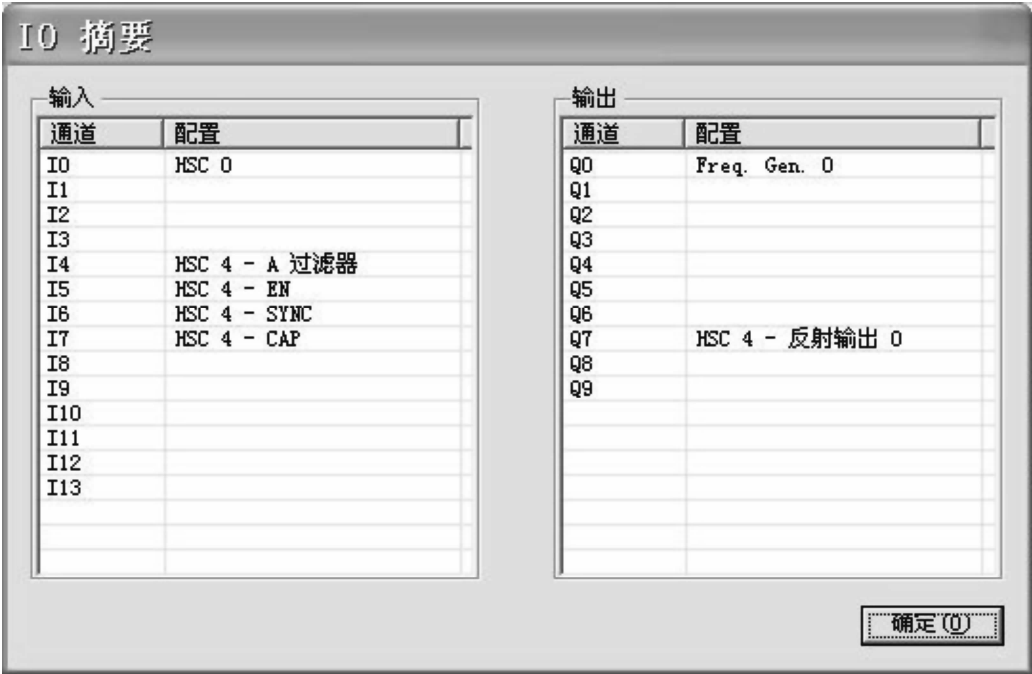


图 17-7　主要计数模式下占用的 IO 点

这些被占用的输入点，诸如 EN、SYNC、CAP 等的作用，我们会在下节来说明。

17.2　计数模式

讨论计数模式前，先让我们认识一些计数概念。

● A 单路脉冲。

● 只有一路脉冲 A 用于计数，如图 17-8 所示。

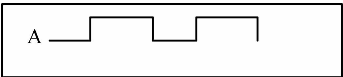


图 17-8　单路脉冲

● A 通道来脉冲，计数上升，B 通道来脉冲，计数下降，如图 17-9 所示。

● 脉冲输入 A 通道→计数上升。

● 脉冲输入 B 通道→计数下降。

● A 脉冲，B 计数方向，如图 17-10 所示。

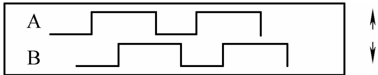


图 17-9　双路脉冲计数

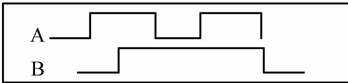


图 17-10　一路脉冲计数，一路控制计数方向

● 脉冲输入 A 通道→计数。

B 通道低电平或高电平 →控制计数上升或下降。

● 倍频模式支持 1 倍、2 倍、4 倍频，如图 17-11 所示。
● X1-CHA 进入脉冲，上升沿计数。CHB = 计数升/降（1 倍频）。
● X2- CHA 进入的脉冲，上升/下降沿都计数。CHB = 计数方向（2 倍频）。
● X4-进入 CHA 和 CHB 的脉冲上升沿和下降沿都计数（4 倍频）。

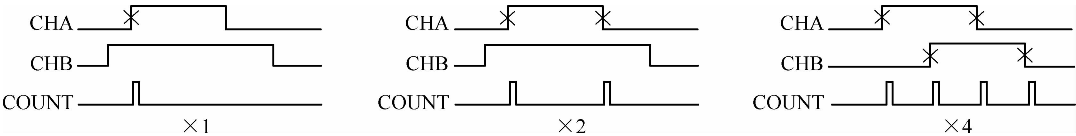


图 17-11　倍频计数

有了这些概念后，我们来看一下计数模式的定义，如图 17-12 所示。

计数器模式	描述	原理
一次性(下降计数)	随着每一次脉冲 A输入，计数器计数值递减。计数的最大范围为31位	A
模数回路 上升/下降计数	随着每一次脉冲A输入，计数器计数值递增。而每一次脉冲B输入，计数器计数值递减。当达到设定值模数后，计数值自动回零，重新开始计数。模数最大范围31位	A B
自由大型计数	计数器动作类似标准上升 /下降计数器，允许负值。计数范围31位加一个符号位	A B
事件计数	计数器允许在给定周期内记录事件个数。每个周期结束，计数器刷新。	A Nbr event
频率计	计数器允许测量频率。即每秒输入的脉冲数。Hz	A Nbr/ s :Hz

图 17-12　计数模式的定义

下面就来详细阐述一下各个计数模式的工作过程。

一次性计数模式的应用如图 17-13 所示。

图中 Preset Value 为预设值，Counter Run 为计数器工作。

步骤 1：IN_SYNC 上升沿触发把预设值调入计数器。

步骤 2：IN_A 口每进入一个脉冲，计数器当前值递减 1，一直到计数器计数值递减为“0”，计数器停止工作。

步骤 3：计数器等待 IN_SYNC 信号的上升沿以便开始新的计数。在这期间,即使 IN_A 口有脉冲进入，计数器也不工作。

步骤 4：如果 EN_Sync 没有启用,IN_SYNC 也不会起作用。EN_Sync 并不是一个信号，它是在计数器组态时激活 IN_ SYNC 的。

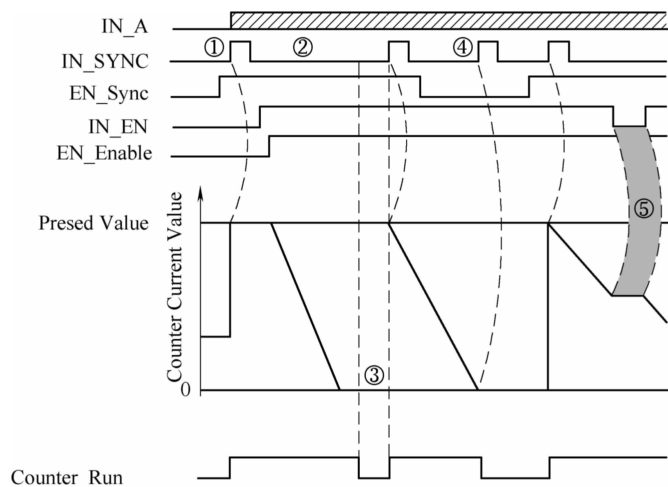


图 17-13　一次性计数模式的应用

步骤 5：在计数器组态时，启用 EN_Enable，则激活信号 IN_EN。IN_EN 为低电平，停止计数器计数，并保持最后的计数值，为高电平后，则继续计数。

模数回路计数模式的应用如图 17-14 所示。

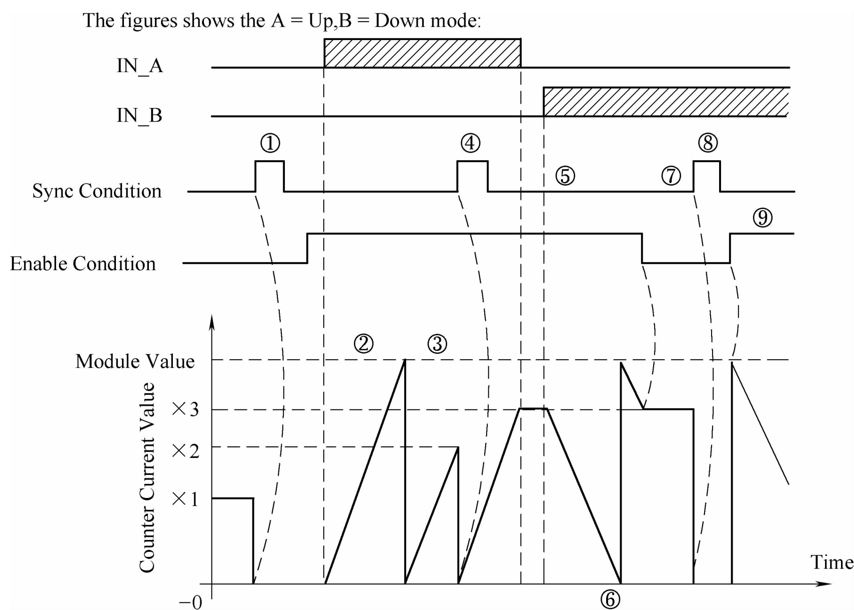


图 17-14　模数回路计数模式的应用

图中，IN_A 脉冲进，计数值上升，IN_ B 脉冲进，计数值下降。

步骤 1：SYNC 信号上升沿触发计数器值回零。

步骤 2：计数使能后，脉冲由 IN_A 进入，计数值增加，到达模数后，自动清零并重新递增计数。

步骤 3：重新递增计数后，计数器的 SYNC 信号可触发计数器回零。

步骤 4：SYNC 触发回零，IN_A 有脉冲进入，则计数器继续递增计数，直到没有脉冲进入。

步骤 5：脉冲从 IN_B 进入，计数器计数值递减。

步骤 6：脉冲从 IN_B 进入，计数器计数值递减到零后，自动回归到模数值并继续递减。

步骤 7：计数器使能信号消失，计数停止。

步骤 8：使能信号没有时，启动 SYNC 信号可使计数器清零。

步骤 9：脉冲从 IN_B 进入，加入使能信号后，计数值回到模数并递减计数。

自由大型计数模式及计数值捕捉如图 17-15 所示。

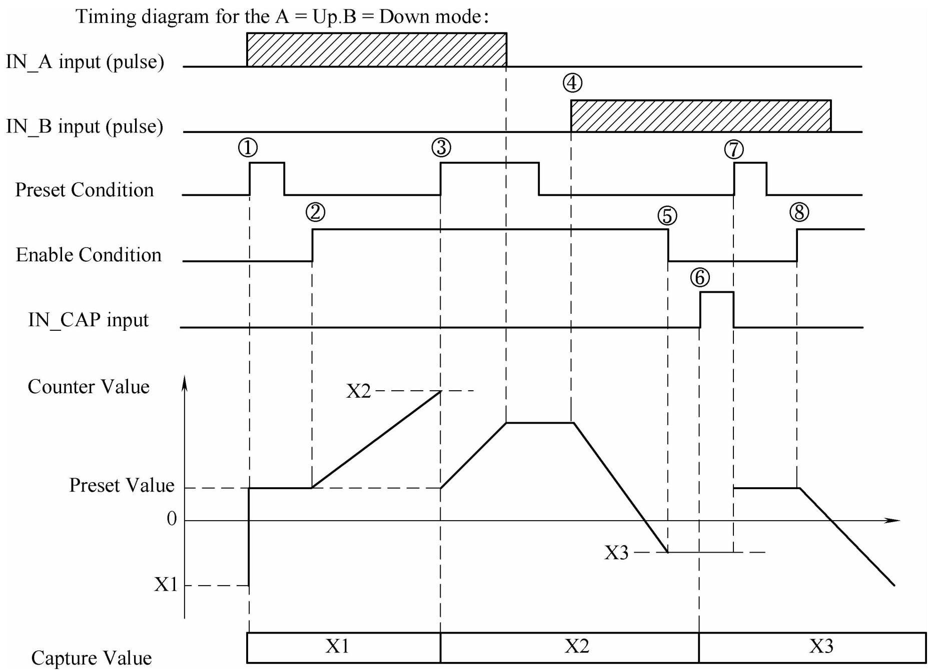


图 17-15　自由大型计数模式及计数值捕捉

图中 IN_CAP 是捕捉信号的输入。Capture Value 是捕捉到的计数值。

步骤 1：预置信号上升沿触发计数器值回到预置值。

步骤 2：计数器使能后，脉冲由 IN_A 进入，计数值增加。

步骤 3：计数值到达 X2 时，预置信号进入把计数值清回到预置值并重新递增，IN_A 没有脉冲进入，计数值停止更新。

步骤 4：IN_B 有脉冲进入，则计数器继续递增计数。

步骤 5：当计数器使能信号消失，计数停止。

步骤 6：捕捉信号进入，把当前计数器计数值捕捉进捕捉值寄存器。

步骤 7：预置信号上升沿触发计数器值回到预置值。

步骤 8：计数器使能后，脉冲由 IN_B 进入，计数值递减，可以到负值。

事件计数模式的应用，如图 17-16 所示。

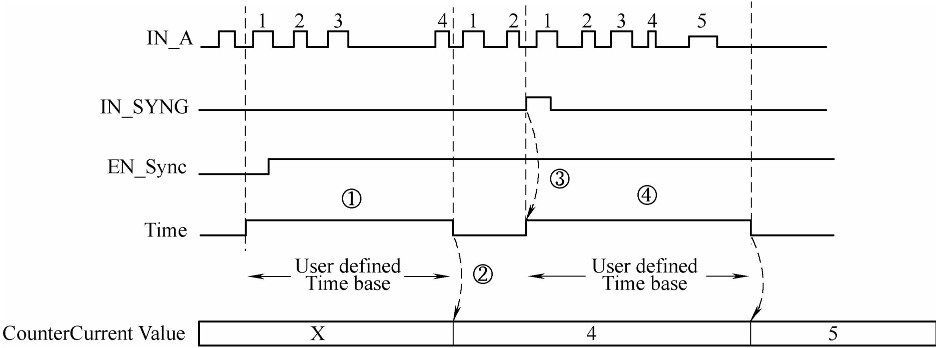


图 17-16　事件计数模式的应用

图中 Counter Current Value 为计数器当前计数值。

从图中，我们可以看到，事件计数器就是用户定义好一个周期，同步信号 IN_SYNC 的进入开启了这个周期的事件计数，周期结束，计数值更新。

频率计模式的应用，用于测量进入 IN_A 的脉冲频率。也就是记录每秒进入 IN_A 口多少个脉冲。最高的频率不能超过 100kHz。

17.3　组态

我们了解了各种计数应用模式和计数器的类型后，就可以根据不同的硬件平台进行组态了。首先要选择的是计数器类型。计数器允许的组态类型有两种：简化/主要。在这两种类型下，可以组态的计数模式如图 17-17 所示。根据这张图，我们可以进行组态来达到用户需求。

计数器模式 \ 输入模式（计数类型，计数模式）	IN_A=脉冲	IN_A=上升 IN_B=下降	IN_A=脉冲 IN_B=方向	倍频
一次性	简化/主要	–	–	–
模数回路	简化/主要	主要	主要	主要
自由-大型	–	主要	主要	主要
事件计数	主要	–	–	–
频率计	主要	–	–	–

图 17-17　高速计数器的组态

首先，在硬件平台上的 HSC 上组态，例如在控制器 M238。

首先在设备树下找到 HSC，并打开 HSC 组态（1），如图 17-18 所示。

然后选择通道，定义计数器类型（2），如图 17-19 所示。

定义好类型，我们就可以进行计数器模式和参数组态，如图 17-20 所示。

如果计数器类型选择了主要，则可以有更多计数模式组态。同时占用的通道也多了。这样可用的计数器个数就大大减少了。

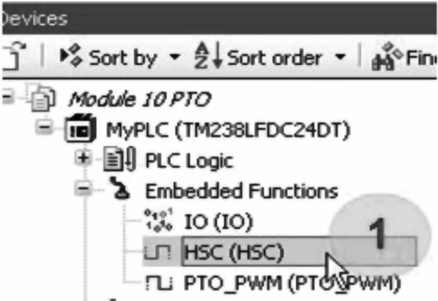


图 17-18　打开高速计数器

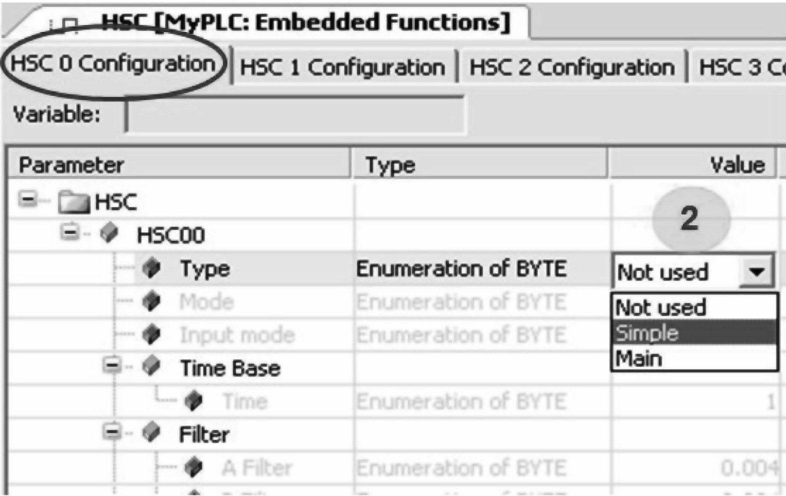
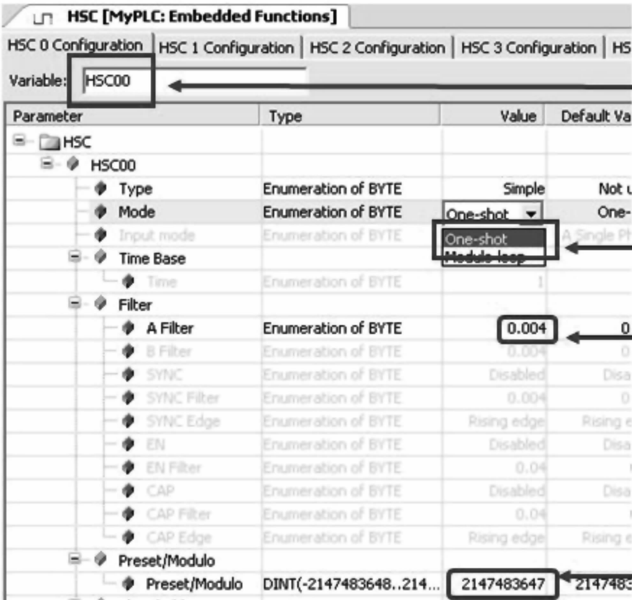


图 17-19　选择通道，定义计数器类型



输入变量名
默认命名HSC00
(第一个0表示是控制器内置，第二个数表示通道)

选择计算模式“One shot”一次性

对进入口A脉冲的滤波
(为了保证脉冲的稳定)

计数器的预置值

图 17-20　简单类型下组态计数模式

17.4　简单和复杂计数的功能块

硬件组态完毕，相应的操控就可以采用计数功能块来实现计数器功能。功能块根据计数器类型也分为简化和主要。

HSC 简化-简单脉冲计数器，功能块如图 17-21 所示。功能块的命名是在组态 HSC 通道时定义好的。因此，这个功能块不能随意命名。先写出字头 HSC，后跟一个点“.”则组态好的计数器名就会出现。如果没组态，则名字就不会出现。

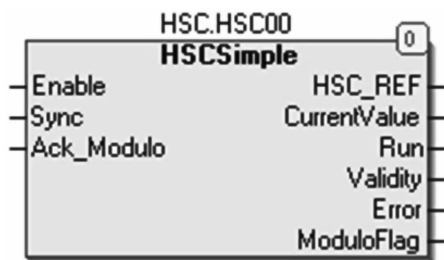


图 17-21　HSC 简化类型功能块

输入描述

1）使能计数器。

2）预置和启动计数。

3）上升沿复位模数到位标志。

输出描述

1）定义连接这个块的高速计数器名称，以便其他功能块调用，比如这个计数器的参数读写。

2）计数器当前值。

3）计数器运行。

4）计数器值有效。

5）错误状态。

6）计数器达到模数值变高。

这个功能块定义了一个和硬件关联的 HSC，在程序中就可以使用了。但要读出和写入计数器的相关参数时，就需要读写功能块了。

HSCGetParam-读高速计数器参数，如图 17-22 所示。这个功能块是可以随意命名的。修改哪个计数器就是由简化或主要计数器输出 HSC_REF 连接的计数器名称决定。

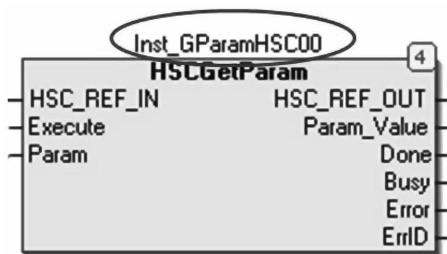


图 17-22　读计数器参数

功能块输入定义

1）定义连接的高速计数器名。

2）上升沿执行功能块。

3）要读出的参数类型（预置值、模数、…），这是一个结构变量。可以查找在线帮助。

输出定义

1）定义高速计数器名称。

2）读出的参数。

3）参数有效。

4）运行中。

5）错误状态。

6）错误代码。

同样，我们也可以对计数器参数进行写入。在这里要写入的参数变为输入值，如图 17-23 所示。

HSC 主要—复杂计数功能块，它有更多的输入和输出，如图 17-24 所示。

例如在主要计数器类型下，我们选择一次性，如图 17-25 所示。

One-shot Main（一次性　主要）– 用于比较复杂应用，占用多个输入。

● 输入/输出在系统中的配置，如图 17-26 所示。

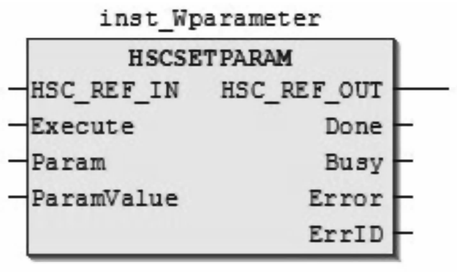


图 17-23　写计数器参数

● SYNC（12）同步计数器的信号；

● Reflex0，Reflex1-输出 Q4/Q5 由系统自动配置给反射输出→因此，这个输出点不要在程序中占用。

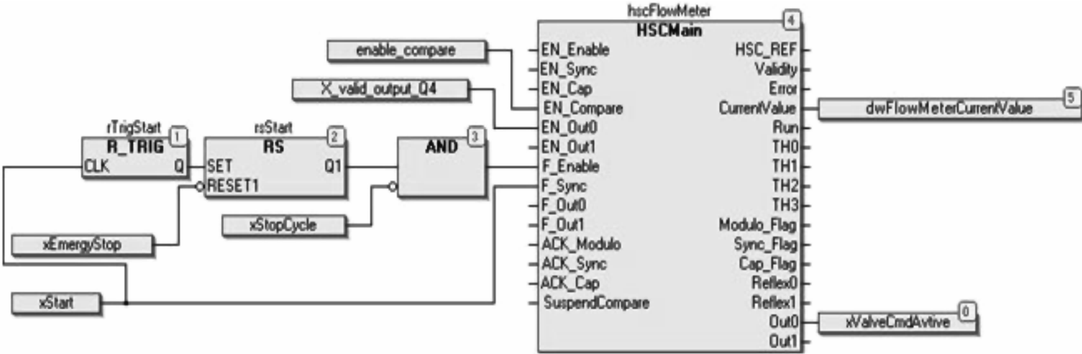


图 17-24　主要计数类型

Parameter	Type	Value
HSC		
HSC00		
Type	Enumeration of BYTE	Main
Parameters		
Mode	Enumeration of BYTE	One-shot
Preset/Modulo	DINT (0...2147483647)	50000
Time	Enumeration of BYTE	1

图 17-25　组态主要计数器

主要计数器-阈值，激活阈值可以触发反射输出，如图 17-27 所示。

设定阈值可以控制反射点的输出位置。

阈值可设定，常用于；

● 在计数范围内触发事件（1）（POU 里执行）；

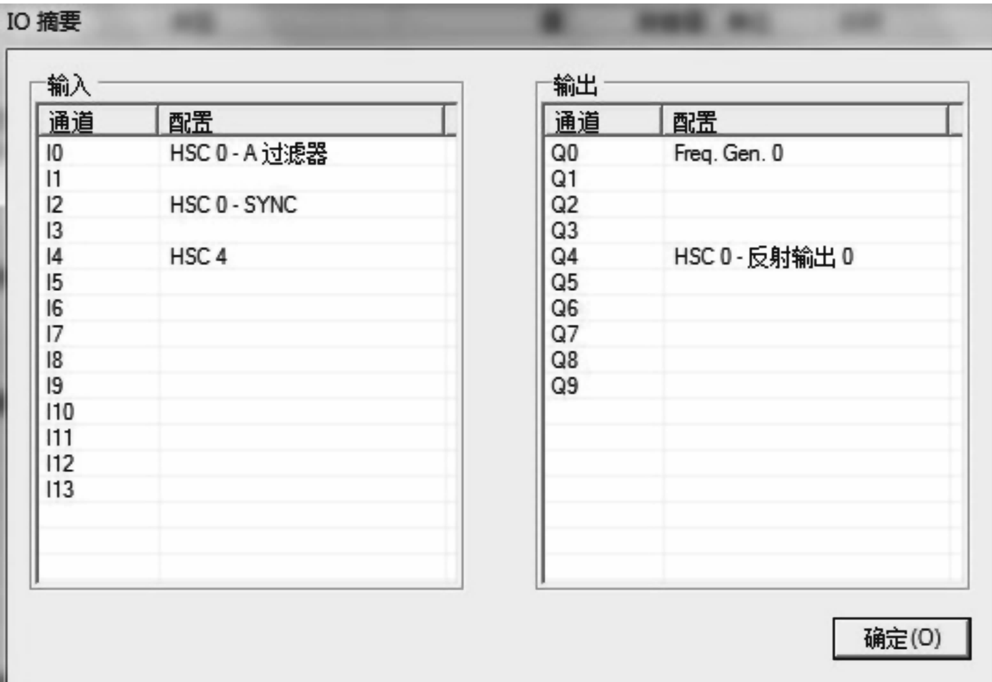


图 17-26　I/O 的配置

- 控制反射输出；
- 每个计数器最多有 4 个阈值。

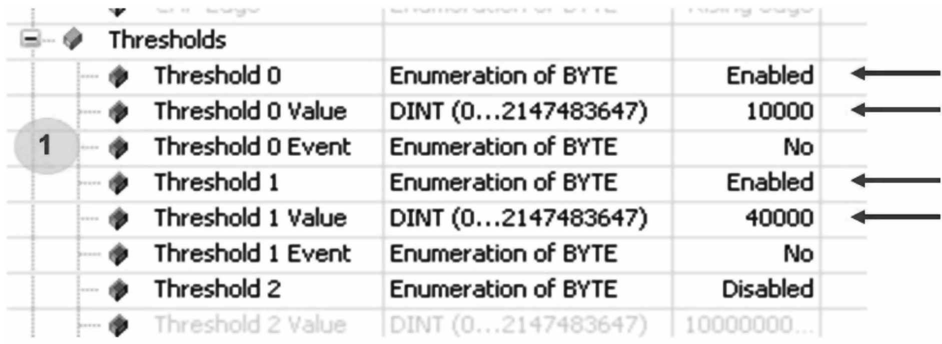


图 17-27　设定阈值的使用

主要计数器-反射动作

激活阈值后，我们就可以定义反射输出的条件，如图 17-28 所示。

- 反射输出 – 可定义 3 种组态方式：
- 计数值 < 阈值 0；
- 阈值 0≤计数值≤阈值 1；
- 阈值 1≤计数值 < 阈值 2。
- 反射输出 0 = Q4。

Reflex Outputs				
Reflex Ouput 0	Enumeration of BYTE	No	No	Set reflex output 0 if value < Threshold 0
Reflex Ouput 0	Enumeration of BYTE	Ye	No	Set reflex output 0 if Threshold 0 <= value < Threshold 1
Reflex Ouput 0	Enumeration of BYTE	No	No	Set reflex output 0 if Threshold 1 <= value < Threshold 2
Reflex Ouput 0	Enumeration of BYTE	No	No	Set reflex output 0 if Threshold 2 <= value < Threshold 3
Reflex Ouput 0	Enumeration of BYTE	No	No	Set reflex output 0 if value >= Threshold 3
Reflex Ouput 1	Enumeration of BYTE	No	No	Set reflex output 1 if value < Threshold 0
Reflex Ouput 1	Enumeration of BYTE	No	No	Set reflex output 1 if Threshold 0 <= value < Threshold 1
Reflex Ouput 1	Enumeration of BYTE	No	No	Set reflex output 1 if Threshold 1 <= value < Threshold 2

图 17-28　反射输出的组态

- 反射输出 1 = Q5。

根据这个定义，在图 17-28 所示的阈值组态中，我们知道，当阈值 0≤计数值 < 阈值 1 时，反射输出 0 即 Q4 有输出。

17.5　PTO-脉冲串输出

在设备树中找到 PTO 的组态，并且在输出模式上选择频率发生器，如图 17-29 所示。

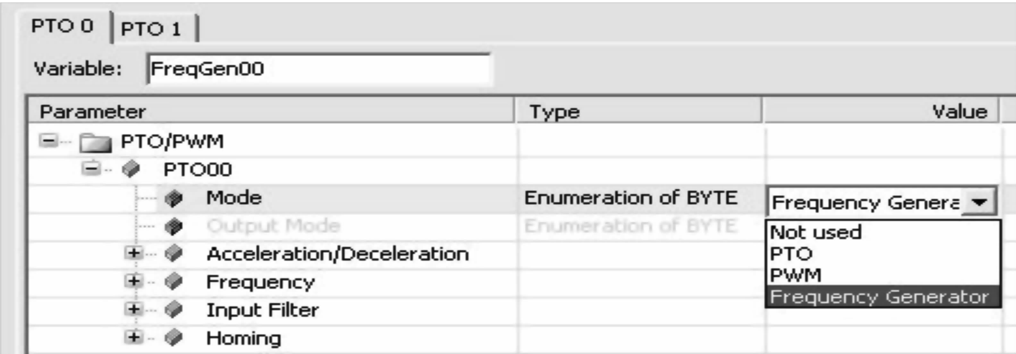


图 17-29　在 PTO 中的组态

在控制器中有 2 路 PTO（Pulse Train Out）内置输出，它的功能如下：

- 具有频率发生器功能；
- 命名的变量作为功能块的后缀名；
- 编辑功能块来产生脉冲；
- 物理输出的实点-快速输出点 Q0 和 Q1。

频率发生器功能块，如图 17-30 所示。

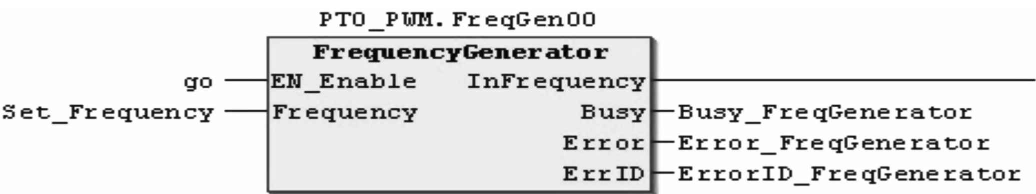


图 17-30　频率发生器功能块

功能块变量名-功能块变量名来自 PTO 0 的组态，把字头 PTO_PWM 写上后，在其后加上点“.”则组态好的名称自动显示出来。如果没组态，则这个名称就显示不出来。

输入描述：

- Enable-使能功能块；
- Frequency-设定产生的频率值。

输出描述：

- InFrequency-达到设定频率输出；
- Busy-发生器工作中；
- Error-出错；
- ErrID-错误代码。

17.6　案例

在一个涂胶机的控制系统中，要求传感器测到工件后，对工件进行涂胶作业。硬件配置和工艺曲线如图 17-31 所示。

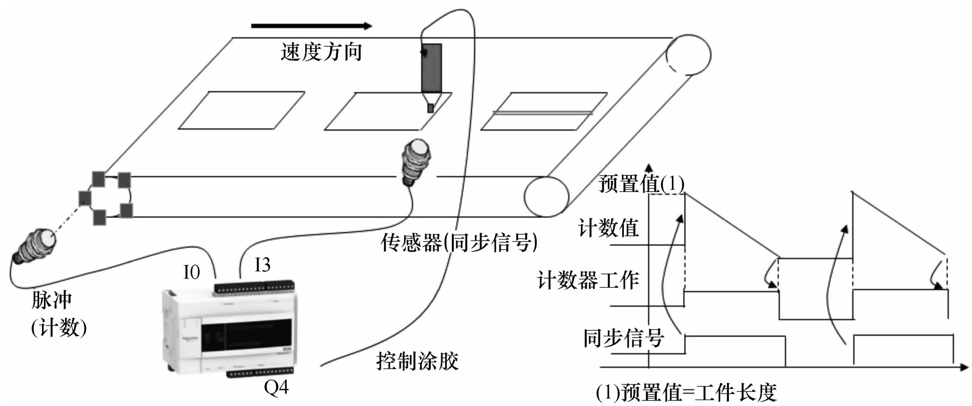


图 17-31　硬件配置和工艺曲线

打开高速计数器组态，如图 17-32 所示。



图 17-32　打开高速计数器组态

打开 IO 配置，定义输入/输出变量如图 17-33 所示。

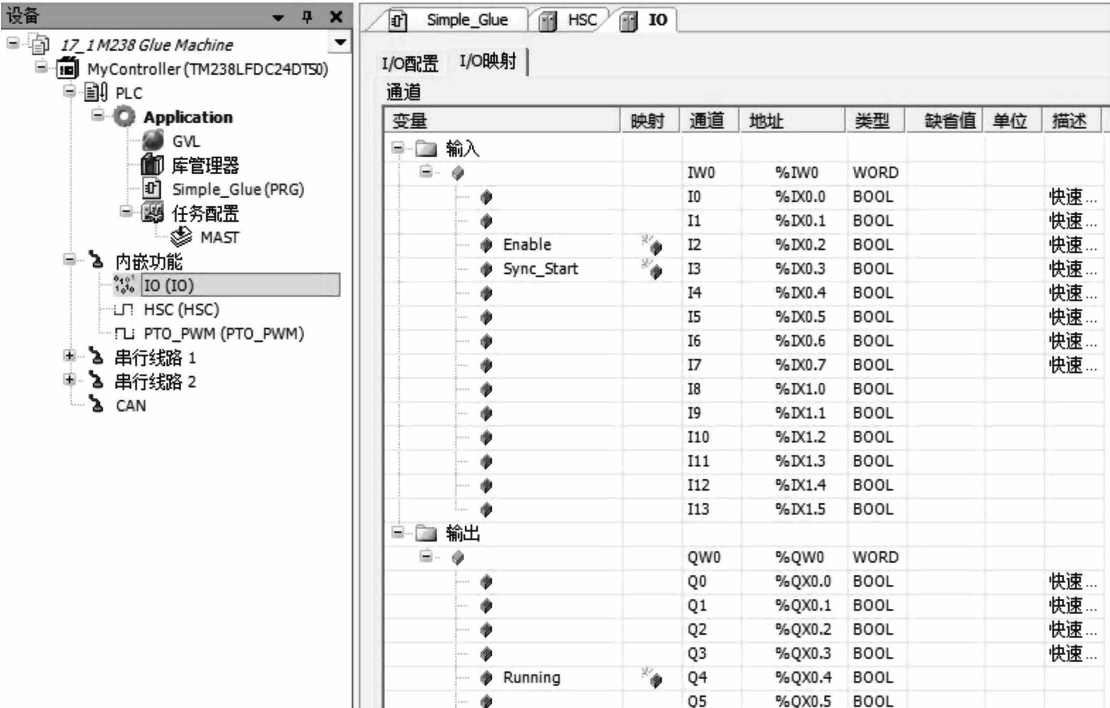


图 17-33　定义 IO 的输入/输出变量

编制的程序如图 17-34 所示。

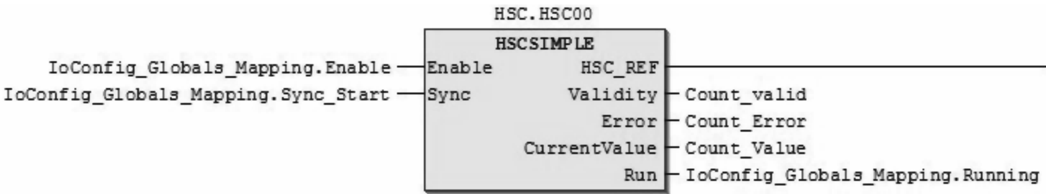


图 17-34　编制的程序

根据计数功能块，I2 高电平使能了计数器功能块。当同步信号来临时，把预设值调入计数器，计数器工作。随着脉冲进入，计数值递减，减到零后，计数器停止工作，等待下一次同步信号的到来，再重新工作。所以，这个涂胶机是把胶涂在整个工件上。

在另一种涂胶机的工作中，用户的要求是把胶涂在工件的一定范围上，如图 17-35 所示。这种涂胶方式就用到了反射输出。

在 HSC 组态，我们采用了计数器的主要类型，选择的计数模式仍然是一次性。预设值为 50000，并且启用同步信号和阈值。主要计数器的组态如图 17-36 所示。

启用阈值后，根据工艺要求，胶要涂在工件的中间部位。即如果工件长度为 50000，那么在 0～20000 这个区间不能涂胶，在 30000～50000 区间也不能涂胶。因此，选择的阈值和反射输出如图 17-37 所示。

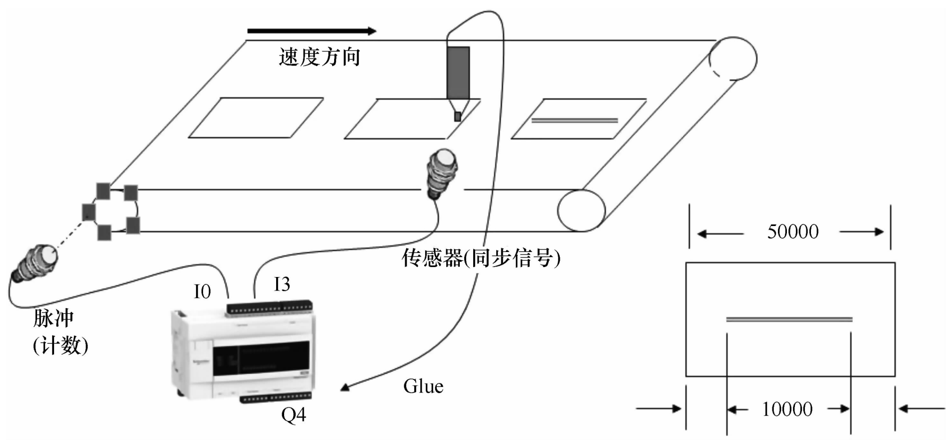


图 17-35　在工件的一部分涂胶

设备
17_2 M238 Glue Machine Evolution
MyController (TM238LFDC24DT50)
PLC
Application
GVL
库管理器
Main_Glue (PRG)
Simple_Glue (PRG)
任务配置
内嵌功能
IO (IO)
HSC (HSC)
PTO_PWM (PTO_PWM)
串行线路 1
串行线路 2
CAN

Main_Glue　HSC

HSC 0 | HSC 1 | HSC 2 | HSC 3 | HSC 4 | HSC 5 | HSC 6 | HSC 7

变量：HSC00

参数	类型	值	缺省值	单位	说明
HSC00					
类型	Enumeration of BYTE	主要	未使用		计数器类型
参数					
模式	Enumeration of BYTE	一次性	一次性		计数模式
预设/模数	DINT (0...2147483647)	50000	2147483647		根据计数模式的预设值或模数值
时间	Enumeration of BYTE	1	1	s	时基
时钟输入					
输入模式	Enumeration of BYTE	单相	单相		输入模式
A 过滤器	Enumeration of BYTE	0.004	0.004	毫秒	过滤值减少了输入计数上的跳动影响
B 过滤器	Enumeration of BYTE	0.004	0.004	毫秒	过滤值减少了输入计数上的跳动影响
辅助输入					
SYNC	Enumeration of BYTE	启用	禁用		启用 SYNC 输入
SYNC 过滤器	Enumeration of BYTE	0.004	0.004	毫秒	过滤值减少了同步输入上的跳动影响
SYNC 沿	Enumeration of BYTE	上升沿	上升沿		SYNC 信号检测（如果未禁用 SYNC 输入）
EN	Enumeration of BYTE	禁用	禁用		启用 ENABLE 输入
EN 过滤器	Enumeration of BYTE	0.04	0.04	毫秒	过滤值减少了启用输入上的跳动影响
CAP	Enumeration of BYTE	禁用	禁用		启用捕获输入
CAP 过滤器	Enumeration of BYTE	0.04	0.04	毫秒	过滤值减少了捕获输入上的跳动影响
CAP 沿	Enumeration of BYTE	上升沿	上升沿		CAP 信号检测（如果未禁用 CAP 输入）
阈值					
阈值 0	Enumeration of BYTE	启用	禁用		阈值激活
阈值 0 值	DINT (0...2147483647)	10000	0		阈值 0 值
阈值 0 事件	Enumeration of BYTE	无	无		在阈值的特定交叉上触发事件
阈值 1	Enumeration of BYTE	启用	禁用		阈值激活
阈值 1 值	DINT (0...2147483647)	40000	500000000		阈值 1 值

Modifiable by programming ◆ = 是 ◆ = 没有

图 17-36　主要计数器的组态

反射输出占用的输出点为 Q4，IO 的组态如图 17-38 所示。

组态完成后，建立与硬件关联的计数器功能块，如图 17-39 所示。

这个功能块使能同步后，一旦探测到工件，就把预设值调入计数器，随着脉冲进入，计数值递减，当计数值递减到 30000 以下时，反射输出 Q4 动作。当计数器值递减到 20000 以下时，反射输出关断，涂胶停止。

HSC 0 | HSC 1 | HSC 2 | HSC 3 | HSC 4 | HSC 5 | HSC 6 | HSC 7

变量： HSC00

参数	类型	值	缺省值	单位	说明
CAP 沿	Enumeration of BYTE	上升沿	上升沿		CAP 信号检测（如果未禁用 CAP 输入）
阈值					
阈值 0	Enumeration of BYTE	启用	禁用		阈值激活
阈值 0 值	DINT (0...2147483647)	20000	0		阈值 0 值
阈值 0 事件	Enumeration of BYTE	无	无		在阈值的特定交叉上触发事件
阈值 1	Enumeration of BYTE	启用	禁用		阈值激活
阈值 1 值	DINT (0...2147483647)	30000	500000000		阈值 1 值
阈值 1 事件	Enumeration of BYTE	无	无		在阈值的特定交叉上触发事件
阈值 2	Enumeration of BYTE	禁用	禁用		阈值激活
阈值 2 值	DINT (0...2147483647)	1000000000	1000000000		阈值 2 值
阈值 2 事件	Enumeration of BYTE	无	无		在阈值的特定交叉上触发事件
阈值 3	Enumeration of BYTE	禁用	禁用		阈值激活
阈值 3 值	DINT (0...2147483647)	2147483647	2147483647		阈值 3 值
阈值 3 事件	Enumeration of BYTE	无	无		在阈值的特定交叉上触发事件
反射输出					
反射输出 0	Enumeration of BYTE	无	无		如果值 < 阈值 0，则设置反射输出 0
反射输出 0	Enumeration of BYTE	有	无		如果阈值 0 <= 值 < 阈值 1，则设置反射输出...
反射输出 0	Enumeration of BYTE	无	无		如果阈值 1 <= 值 < 阈值 2，则设置反射输出...
反射输出 0	Enumeration of BYTE	无	无		如果阈值 2 <= 值 < 阈值 3，则设置反射输出...

图 17-37　阈值的设定及反射输出的组态

IO 摘要

输入

通道	配置
I0	HSC 0 - A 过滤器
I1	
I2	HSC 0 - SYNC
I3	
I4	
I5	
I6	
I7	
I8	
I9	
I10	
I11	
I12	
I13	

输出

通道	配置
Q0	Freq. Gen. 0
Q1	
Q2	
Q3	
Q4	HSC 0 - 反射输出 0
Q5	
Q6	
Q7	
Q8	
Q9	

确定(O)

图 17-38　I/O 的组态

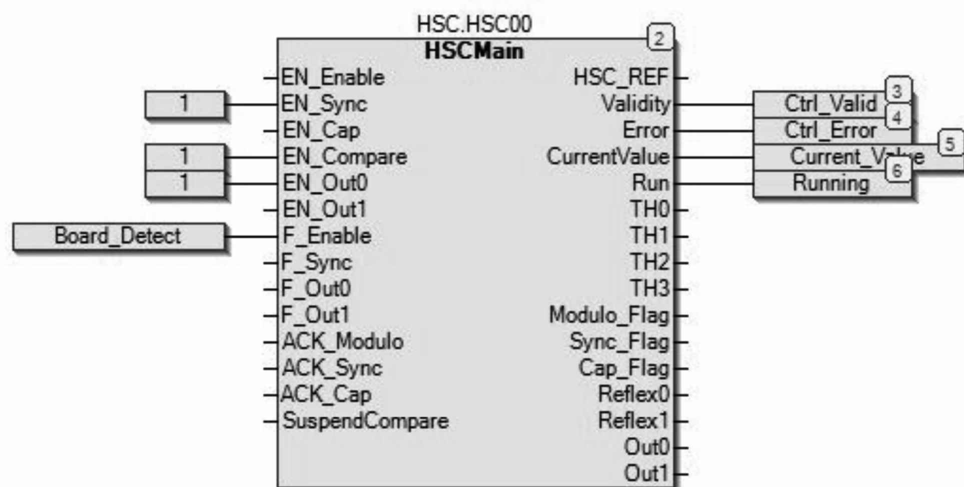
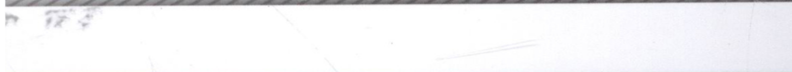
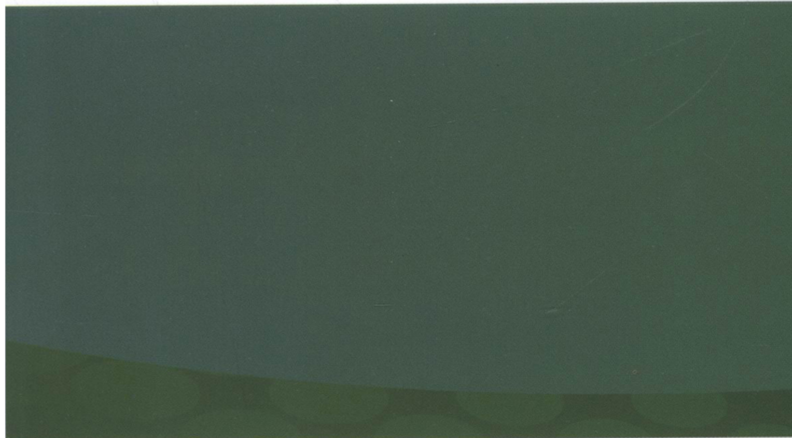


图 17-39　主要计数器的功能块





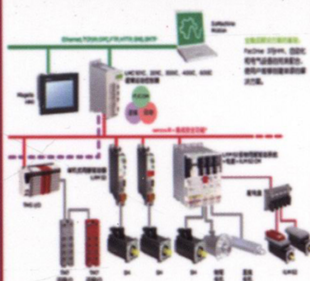


各个行业的自动化解决方案

OEM
解决方案

终端用户
解决方案





地址：北京市百万庄大街22号
邮政编码：100037
电话服务
社服务中心：010-88361066
销售一部：010-68326294
销售二部：010-88379649
读者购书热线：010-88379203
网络服务
教材网：http://www.cmpedu.com
机工官网：http://www.cmpbook.com
机工官博：http://weibo.com/cmp1952
封面无防伪标均为盗版

上架指导 工业技术／工业自动化

ISBN 978-7-111-46531-7
ISBN 978-7-89405-375-6(光盘)

策划编辑◎林春泉

ISBN 978-7-111-46531-7



定价：69.00元(含1CD)